Spring 2012

CRYPTSIM: SIMULATORS FOR CLASSIC ROTOR CIPHERS

Miao Ai
San Jose State University

CRYPTSIM: SIMULATORS FOR CLASSIC ROTOR CIPHERS

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Miao Ai

May 2012

The Designated Project Committee Approves the Project Titled

CRYPTSIM: SIMULATORS FOR CLASSIC ROTOR CIPHERS

by

Miao Ai

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2012

Dr. Mark Stamp	Department of Computer Science
Dr. Soon Tee Teoh	Department of Computer Science
Dr. Richard Low	Department of Mathematics

ABSTRACT

CryptSim: Simulators for Classic Rotor Ciphers

by Miao Ai

In this project, web-based visual simulators have been implemented for three classic rotor cipher machines: Enigma, Typex, and Sigaba. Enigma was used by Germany during World War II, while Typex is a British cipher that was based on the commercial version of the Enigma. Sigaba is a relatively complex machine that was used by the Americans during the 1940s and into the 1950s. Sigaba is the most secure of the three ciphers, there was no successful attack on Sigaba during its service lifetime.

Our web-based visual simulators are functionally equivalent to the actual electro-mechanical machines. Each simulator allows the user to initialize the key and encrypt or decrypt. Also, each simulator provides a web-based "play station" that allows the user to understand how these classic ciphers work by observing their internal operations when encrypting and decrypting. These simulators do not require any installation, and users can access the simulators provided they have access to the Internet.

ACKNOWLEDGMENTS

To my mother who raised me to be the person I am today.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

During World War II, a large number of cipher machines were made and used by different countries. For example, Enigma was invented and used by Germany; Typex was developed by the British based on the commercial Enigma; and Sigaba was a superior electro-mechanical, rotor-based cipher machine created by the Americans. These classic ciphers, which once played an important role in history, represented a major achievement of human ingenuity and the technologies they used paved the way for modern computers and cryptographic systems.

1.1 Previous Work

Scholars and students around the world have already done a lot of research on classic ciphers and simulators have been created as well. In 1980 Marian Rejewski published the first paper detailing the Polish cryptanalysts successes in breaking the Enigma [1]. Frank Spiess implemented a Flash simulator that visualizes the three-rotor-Enigma, including German, Polish, Italian and Spanish versions [2]. Wing On Chan and Heather Ellie Kwong both wrote papers about the Sigaba cipher, and described attacks on it [3] [4]. Geoff Sullivan and Frode Weierud have done a large amount of research on various classic ciphers, and Geoff Sullivan implemented a series of visual simulators, including Enigma, Sigaba, etc [5].

1.2 Project Overview

Certain limitations can be found in the previously mentioned simulators. For example, the Flash simulator created by Frank Spiess, only simulates the three-rotor

Enigma, does not include any other classic ciphers; also it does not allow the user to change Enigma's reflector within two alternatives. Another example is the series of visual simulators implemented by Geoff Sullivan. They simulate various classic cipher machines but these simulators can only run on certain Windows platforms and most of those simulators do not have user friendly interfaces for visualizing key setup phase.

In this project our goal is to implement a web-based software that simulates three World War II era classic cipher machines: Enigma, Typex and Sigaba. Since it is a web-based application, user can use it as long as they have access to the Internet. There is no installation and no specific operating system or platform required. This application consists of three visual simulators, each one of them is functionally equivalent to the actual Enigma, Typex or Sigaba used during WWII, respectively. Moreover, this application allows its user to do key initialization by themselves at a web-based "Setup" very easily and makes users understand how these ciphers work by presenting the internal structure and signal journey during encrypt and decrypt operations at a web-based "Play Station".

In this project report, we begin with Chapter 2 by introducing Enigma, Typex and Sigaba. We discuss in detail the backgrounds, cryptographic important components, and explain the working principles for each of them. In Chapter 3 we discuss how we simulate the components, key setup phases, and encrypt, decrypt operations for three ciphers. Chapter 4 concludes the project and indicates future work. The Appendix A, Appendix B, and Appendix C contain example to verify that all three visual simulators produce the correct plaintext and ciphertext.

CHAPTER 2

Background

2.1 Enigma

Enigma was an electro-mechanical rotor-based cipher machine invented by German engineer Arthur Scherbius. As early as the beginning of the 1920s, the Enigma cipher had already been used in the commercial area. The Enigma has continued to evolved since then, and many different versions, with certain common features, have been developed. In the mid-1920s the German government and military started to develop a modified version of the commercial Enigma. Enigma was used by the German military and government before and during World War II [6].

Successful attacks were conducted during Enigma's service time. At the end of the year 1932, the Polish Cipher Bureau broke Germany's military Enigma ciphers for the very first time [7]. From 1938 the Enigma machines were added complexity by Germany continually, making the initial decryption techniques decreasingly successful. However the Polish attacks already provided an important foundation, so the later British cryptanalysts were able to decrypt a large number of messages that encrypted by Enigma. During the war, several different versions of the Enigma were used, but the German Wehrmacht version is the one that is most commonly known by the public, and it is the one that we focus on in this paper [6].

2.1.1 Enigma Cipher Machine Overview

Figure 1 shows an Enigma cipher machine. At the top of the Enigma cipher machine, are three cipher rotors and a reflector. A lightboard is visible in the middle, which is used to indicate the output. A keyboard appears in the front, which is

essentially, a mechanical typewriter. In the front panel, below the keys, an Enigma plugboard, or so called "stecker" is visible.

Figure 1: Enigma cipher [8]

2.1.2 Enigma Cipher Machine Components

In order to understand the working principle of Enigma, we need to get familiar with each important cryptographic component in Enigma and understand how they work. We will introduce three components: cipher rotor, reflector, and stecker.

Enigma Cipher Rotor

The Enigma cipher rotor has 26 electronic contacts on each side, representing alphabets from A to Z. When the very basic Enigma was invented in 1918 by Arthur Scherbius, his idea was to invent a cipher that encrypts a message by performing a number of substitutions one after the other by electrical connections. For example as shown in Figure 2, there is a wire from letter Q in the top row to letter M in the bottom row. Thus a signal can go from letter Q to the letter M [9]. Figure 2 shows which of the 26 wires which will give the effect of substitution.

The two rows of letters in Figure 2 can actually be seen as an individual Enigma

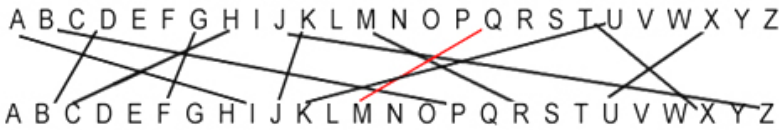

Figure 2: Signal goes through one cipher rotor [9]

cipher rotor: the top row represents 26 contacts on one side of the rotor, and the bottom row represents contacts on the other side. Next, the very natural idea is adding another rotor. As shown in Figure 3, the second rotor is represented by two rows at the bottom. So now the signal goes from letter Q to the letter M at the first rotor, from letter M at the first rotor, it touches the letter M at the second rotor, and goes from letter M to letter R. Figure 3 shows just a few of the 26 wires as well. Using this method, we can compose substitutions which are to be performed one after the other [9].

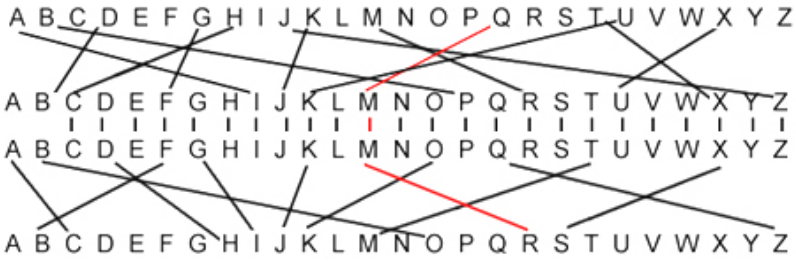

Figure 3: Signal goes through two cipher rotors [9]

If only this strategy was used, Enigma would be nothing more than a mono-alphabetic substitution (simple substitution) cipher, with the initial settings determining the permutation [10]. However, Enigma rotor steps during the operation. For example, in Figure 4, we can see that when the second rotor is displaced by two letters, the substitution is affected by this displacements, and becomes quite different [9]. So now the signal still goes from letter Q to the letter M at the first rotor, but from letter M at the first rotor, it touches the letter K at the second rotor, and goes from letter K to letter J. Figure 4 also only shows fraction of the 26 wires.

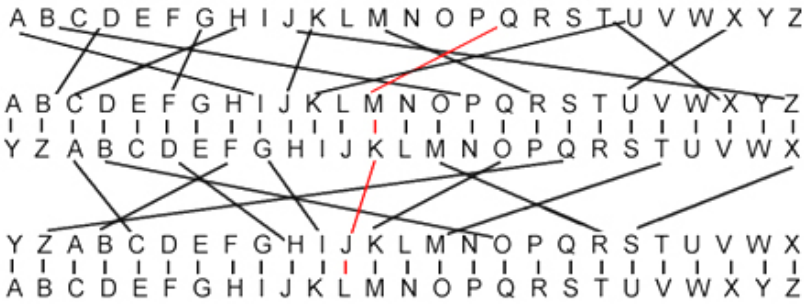

Figure 4: Signal goes through two cipher rotor with displacement [9]

This makes Enigma a poly-alphabetic substitution cipher, which means different substitution alphabet are used for different parts of the message [10]. It also increases the complexity of the Enigma cipher, which resulted in a relatively higher security in its era. However, Enigma still did not achieve the highest possible security design during that time The title of the most secure rotor cipher during WWII belongs to another rotor-based cipher machine: Sigaba. One of the most significant difference between Enigma and Sigaba is that Enigma was using regular stepping, but Sigaba used irregular stepping. More detail about Sigaba will be introduced later.

From Figure 4, there is another thing that needs to be mentioned. The inverse permutation can be obtained by simply passing a signal through letter L at the second rotor, and goes to letter K, touches letter M at the first rotor, and arrives at letter Q. This is a useful feature, since we can decrypt with the same bank of rotors used to encrypt [10].

As we already know from previous introduction, since the development of Enigma, many different versions has emerged. Some Enigma versions, like Kriegsmarine M4, had four cipher rotors, other versions had even more. However, in this project, we focus on the German Wehrmacht version which initially came with three cipher rotors, therefore we assume that Enigma cipher has three cipher rotors [11].

Theoretically, there is no limit if written in software, therefore a huge numbers of rotors with different permutations can be used. But in this project, we assume the Enigma cipher rotor only has five alternatives because this assumption matches what was available in an actual WWII Enigma cipher machine [9]. Table 1 shows all five alternatives.

Rotor	Permutation
Rotor 1	EKMFLGDQVZNTOWYHXUSPAIBRCJ
Rotor 2	AJDKSIRUXBLHWTMCQGZNPYFVOE
Rotor 3	BDFHJLCPRTXVZNYEIWGAKMUSQO
Rotor 4	ESOVPZJAYQUIRHXLNFTGKDCMWB
Rotor 5	VZBRGITYUPSDNHLXAWMJQOFECK

Table 1: Five Enigma cipher rotor alternatives [9]

Enigma's three cipher rotors are marked as rightmost cipher rotor, middle cipher rotor and leftmost cipher rotor, and placed in the cipher rotor bank as the order shown in Figure 5. We use the following notation in Figure 5:

L = leftmost cipher rotor

M = middle cipher rotor

R = rightmost cipher rotor.

Figure 5: Enigma cipher rotor bank

Each Enigma cipher rotor has one or multiple notches, depending on its version. In this project, we focus on the Enigma cipher rotor that has single notch. Notch

is a small peg that attached to the rotors It played an important role on controlling Enigma rotor stepping [12]. For example, if a rotor's notch is Q, then when this rotor steps from Q to R (or from Q to P, depending on its stepping orientation), the next rotor is advanced [13]. Notch can be changed, although they seem not to have been in practice. Table 2 shows default notch setting for Enigma cipher rotors.

Rotor	Notch
Rotor 1	Q
Rotor 2	E
Rotor 3	V
Rotor 4	J
Rotor 5	Z

Table 2: Notch for Enigma cipher rotor [6]

Now recall from our previous analysis that the security of a rotor cipher comes from the usage of multiple rotors and rotor stepping. When using Enigma to encrypt or decrypt, the rightmost rotor steps whenever a key is pressed, and the other rotors step in an odometer-like fashion, that is the middle cipher rotor steps once for each 26 steps of the rightmost cipher rotor and the left rotor steps once for each 26 steps of the middle rotor [10]. The exception of this odometer effect happens when the middle rotor just stepped to the position that when the next key is pressed, the leftmost rotor will be activated to step. From our research, we know that the mechanical system used by Enigma makes whenever a rotor step, it triggers the rotor to its right to also step. Therefore the above exception situation means when the next key is pressed, not only the leftmost rotor steps, the middle rotor steps as well. This makes a "double stepping" happens to the middle rotor in succession. The same mechanical system makes the middle rotor step triggers the rightmost rotor step as well. However since the rightmost rotor already steps with every key pressing, no noticeable effect occurs

on the rightmost rotor [10]. We will give an example about this special occasion when we introduce the Enigma visual simulation at Chapter 3 to verify that our application simulates this as well.

The Enigma cipher's rotors always step in a forward direction, when it steps, it steps alphabetically. For example, like shown in Figure 6, Enigma cipher rotor steps from letter O to letter P, from letter P to letter Q.

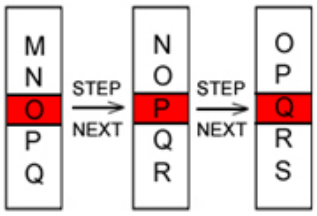

Figure 6: Enigma cipher rotor [10]

Enigma Reflector

Enigma has one reflector, it does not step during encryption or decryption. The Enigma reflector has 26 electronic contacts on just one side, these contacts connect to each other in a scrambled fashion. In a sense, the Enigma reflector can be viewed as a fixed plate that connects letters in pairs. During operation, when an input letter passes through the third cipher rotor and an output is generated, Enigma uses its reflector to process that output. That output is swapped by the reflector, and the new output is then passed back through three cipher rotors in an inverse direction. The above procedure is illustrated in Figure 7.

The reflector adds extra complexity to Enigma but the addition of the reflector made the Enigma machine reciprocal [9]. This means the same reflector can be used to do both encrypt and decrypt operations.

Theoretically, there is also a large number of reflectors with different permuta-

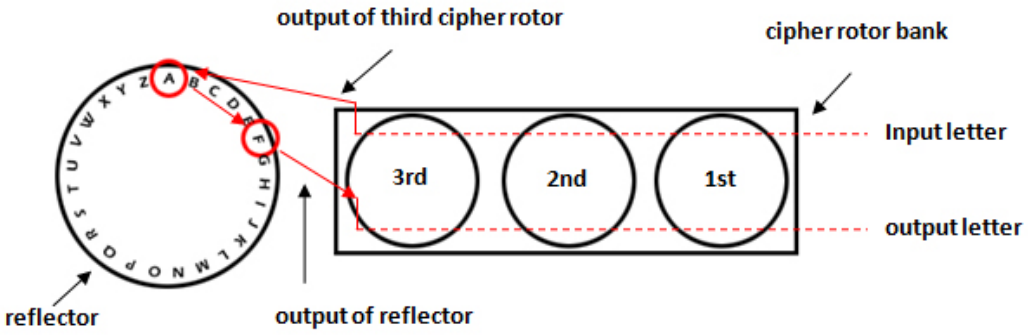

Figure 7: Signal goes through Enigma reflector

tions can be used when we implement Enigma in software. However, in this project, we assume there are only two alternatives due to the limitation the actual Enigma cipher machine had during WWII [9].

Enigma Stecker

Enigma has a plugboard, or "Stecker" in German. It was first introduced on German Army Enigma machines, and adopted by the Navy later. Enigma stecker is essentially a plugboard that allows its user to connect 26 letters in 13 pairs using cables. Each letter on the plugboard has two sockets, the upper socket (from the keyboard) and the lower socket (to the entry-rotor). A cable is used to switch the connections of the two letters: a plug at one end of the cable is inserted to the upper and the lower sockets of a letter to disconnected them; the plug at the other end of the cable is inserted into another letter's sockets. Therefore the connections of the two letters are switched. As illustrated in Figure 8, two pairs of letters have been swapped (A-J and S-O) [6].

Because stecker affected both the incoming signal from the keyboard and the outgoing signal to the lightboard, it left unchanged the reciprocal property of the Enigma [9].

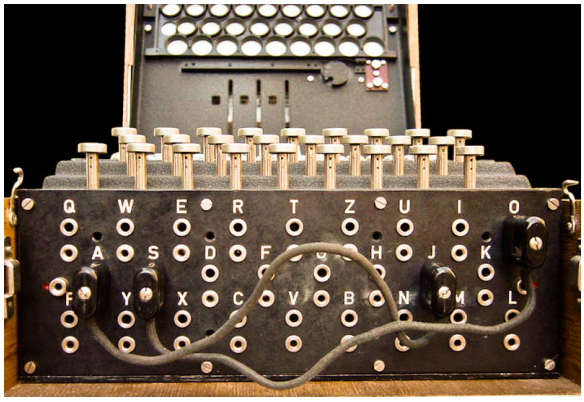

Figure 8: Real WWII Enigma stecker [15]

2.1.3 Enigma Encryption

When the user chooses to do an encrypt operation using Enigma, a key is pressed at the keyboard, and an electronic signal is generated and goes to the stecker. Stecker maps the input letter to another letter if a connection was present during key configuration. The electronic signal then goes to the right side of the cipher rotor bank, goes through each cipher rotor from one side to the other following the direction: the rightmost cipher rotor first, the middle one later, and the leftmost one at last. The electronic signal then goes to a contact on the reflector, and comes out at a different contact. After it goes out of the reflector, the electronic signal then goes back to the cipher rotor bank, starts from the the leftmost rotor this time, and goes through three rotors again. The electronic signal then reaches the stecker again, the output letter is re-mapped back to another letter if a connection was made and lights the corresponding bubble up on the lightboard. This electronic signal journey is illustrated at Figure 9.

At key setup phase, there are three steps are needed before Enigma performs an encrypt operation:

a. Set up three Enigma cipher rotors

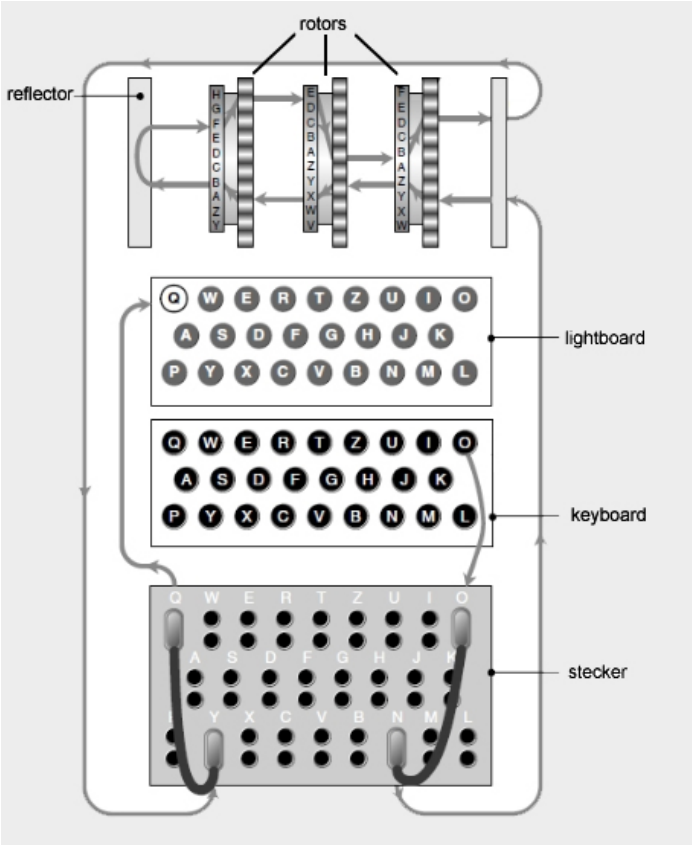

Figure 9: Enigma diagram [16]

- Choose three cipher rotors out of five alternatives
- Decide the order of these three rotors
- Set starting position for these three rotors

b. Set up Enigma reflector

- Choose one reflector out of two alternatives

c. Set up Enigma stecker

- Wire letters into pairs

2.1.4 Enigma Decryption

As we already know, because of Enigma's reciprocal property, when we use Enigma to do a decrypt operation, we can use the same hardware with exactly the same settings that used to do an encrypt operation [14]. So when user types in the ciphertext, he/she will receive the plaintext as the output.

2.2 Typex

Like Enigma, Typex was also an electro-mechanical, rotor-based cipher machine, but used by the British armed force [17]. Typex was an adaptation of the commercial German Enigma with several enhancements that increased its security [18].

In the 1920s, British government started to look for a replacement for their existing book code systems. Later, in 1926, an inter-departmental committee was established to be responsible on cipher machines. In 1934, Group Captain O. G. Lywood set about building a copy of the Enigma using mainly parts from commercial teletypes and made it serve RAF (British Royal Air Force) [17]. Compared to Enigma, Lywood's machine was much bigger and heavier. The resulting machine was initially named the "RAF Enigma with Type X attachments" [18]. In early 1937, 29 Typex Mark I cipher machines were made according Lywood's prototype and used to equip the main RAF headquarters. Later, a number of improvements were made on Typex Mark I, and a much better machine had been produced, this machine was known by people as Typex Mark II. In June 1938, the committee approved an order of three hundred and fifty Typex Mark II cipher machines. The later Typex Mark III was hand operated, although still more cumbersome than Enigma, it was much smaller and lighter than its predecessors. Typex Mark VI was also a hand operated variant, the later Typex Mark 22 and Typex Mark 23 were essentially Typex Mark VI with

modifications. For example, both Typex 22 and Typex 23 incorporated plugboard. Typex Mark VIII was the first model that can interface with other Typex machines by transmitting Morse code [17].

2.2.1 Typex Cipher Machine Overview

Figure 10 shows a Typex cipher machine. At the top of Typex cipher machine, are five rotors, and one reflector. The two rotors at right hand side are stators, the other three are cipher rotors. A keyboard appears in the front. Unlike Enigma, Typex does not have stecker or plugboard in its front panel.

Figure 10: Typex diagram [19]

2.2.2 Typex Cipher Machine Components

From previous introduction, we know that Typex was a copy of, or at most, an adaptation of Enigma. Therefore most components and the basic working principles of Typex are quite similar to Enigma. We already discuss Enigma components from previous introduction, so we only describe the difference here.

Typex Cipher Rotor and Stator

Typex cipher rotor have the same structure and working principle as Enigma

cipher rotor. Therefore the Typex cipher rotor and the Enigma cipher rotor use the same visualization. Typex stator has different usage, but it is the same rotor as the Typex cipher rotor, therefore they are interchangeable, selected from the same set of alternatives, and use the same visualization as well. Details about visualization will be introduced in the next chapter. In this project, we assume there are eight alternatives for Typex, because of the limitation that a real Typex had during WWII. Table 3 shows all eight alternatives.

Rotor	Permutation
Rotor 1	QWECYJIBFKMLTVZPOHUDGNRSXA
Rotor 2	AJDKSIRUXBLHWTMCQGZNPYFVOE
Rotor 3	BDFHJLCPRTXVZNYEIWGAKMUSQO
Rotor 4	ESOVPZJAYQUIRHXLNFTGKDCMWB
Rotor 5	VZBRGITYUPSDNHLXAWMJQOFECK
Rotor 6	FVPJIAOYEDRZXWGCTKUQSBNMHL
Rotor 7	KZGLIUCJEHADXRYWVTNSFQPMOB
Rotor 8	ZLVGOIFTYWUEPMABNCXRQSDKHJ

Table 3: Eight Typex cipher rotor alternatives [20]

The three Typex cipher rotors are placed in the Typex rotor bank the same order as Enigma, and they are also marked as rightmost cipher rotor, middle cipher rotor and leftmost cipher rotor. The two stators are placed next to the Typex cipher rotors at right side. Figure 11 shows the positions of all five rotors. We use the following notation in Figure 11:

L = leftmost cipher rotor

M = middle cipher rotor

R = rightmost cipher rotor

LS = left stator rotor

RS = right stator rotor.

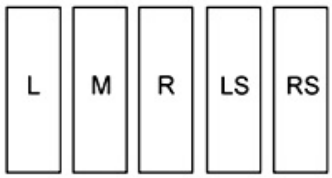

Figure 11: Typex rotor bank

However, unlike the Enigma cipher rotor, Typex cipher rotors can be placed in both forward and backward directions. If the Typex cipher rotor is inserted and steps in forward direction, it steps anti-alphabetically as illustrated in Figure 12.

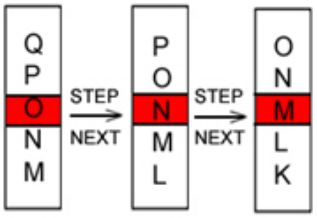

Figure 12: Typex rotor in forward orientation [12]

But if the Typex cipher rotor is inserted and steps in backward direction, it steps alphabetically with letters appearing upside down on the rotors as illustrated in Figure 13 [10].

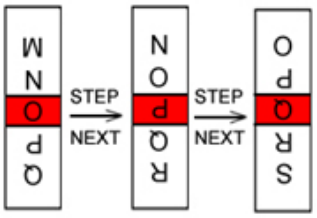

Figure 13: Typex rotor in reverse orientation [12]

Typex stators can be inserted in both forward and reverse directions as well, but unlike Type cipher rotors, they do not step during encrypt and decrypt operations. Typex stators provide a similar protection to Enigma stecker, but Typex stators do not have the same reciprocal property as Enigma stecker [12].

Typex rotors have multiple notches, this is different from Enigma rotor as well. The number of notches can be five, seven or nine, this means the middle and left rotors would turn over multiple times for each rotation to its left neighboring rotor. In use, notches for each rotor were the same [21]. There is no need for unique notch settings for each rotor, one notch setting is used by all rotors to determine whether to step or not based on its current position. In this project, we focus on a nine notch situation, which are A, C, E, I, N, Q, T, V, and Y [20].

Typex Reflector

Like Enigma, there is potentially a huge number of alternative reflectors that can be used when implement in software. However in this project, we assume there is only one reflector available for Typex, this matches the limitation of a real Typex cipher had [20]. The Typex reflector has the same structure as Enigma reflector: 26 electronic contacts on one side, and they have the same working principle. Like the Enigma reflector, Typex reflector also does not step during encrypt and decrypt operations.

2.2.3 Typex Encryption

When a Typex user chooses to encrypt using Typex, a key is pressed at keyboard, and an electronic signal is activated and is sent to Typex. This electronic signal goes to the right stator first, and then to the left one. The signal then goes through three cipher rotors in a right to left direction, followed by crossing the reflector. After comes out of the reflector, the signal then goes through three cipher rotors again in a left to right direction this time. The signal then reaches two stators and goes through them, left one first, right one later, and the output is printed out from there. This signal journey is illustrated at Figure 14.

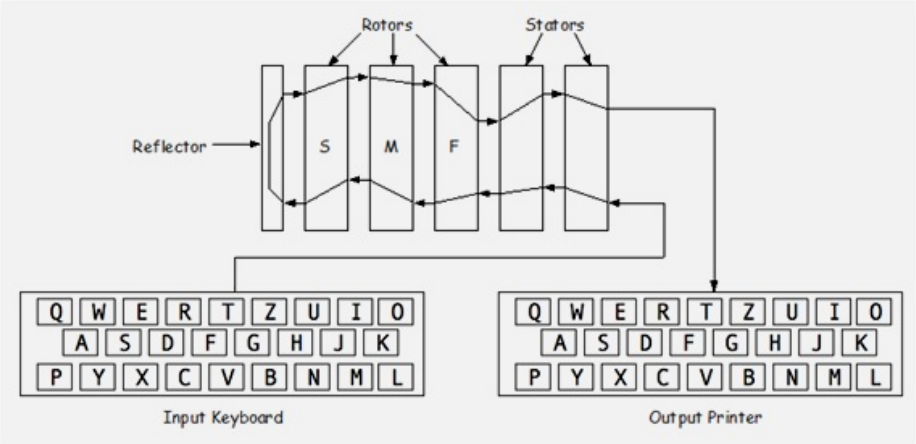

Figure 14: Typex diagram [12]

One more thing that needs to be mentioned is that when an encryption operation is performed with Typex, user can type the space key, and the space key is changed to X prior to enciphering [21].

At key setup phase, there are four steps are needed before using Typex to do encryption:

a. Choose five rotors out of eight alternatives, three of them are cipher rotors and the left two are stators

b. Decide order for cipher rotors and stators

c. Decide orientations for cipher rotors and stators

d. Choose starting position for cipher rotors and stators

Since we assume that the Typex reflector has no alternatives, at key setup phase, the user does not need to do specific setting up for it. Our web-based Typex visual simulator adds a reflector automatically.

2.2.4 Typex Decryption

To decrypt, the user sets up the key exactly the same way as encryption. But this time, the user needs to type in ciphertext, and gets plaintext as the decrypted output. Also, when deciphering, output letter X is not changed back to the space key, since both plaintext X and the space key are decrypted as X [12].

2.3 Sigaba

Sigaba is also known as Converter M-134C in US Army, or as CSP-888/889 in US Navy [22] [23]. In addition, the name ECM Mark (Electric Cipher Machine) II was used during the development of the machine [10]. Sigaba was also a rotor-based, electro-mechanical cipher machine, it was developed by Americans in the late 1930, used during WWII and until 1950s. As far as public known, there is no successful attack ever been conducted during Sigaba serving time.

Even before WWII, US cryptographers realized that the regular stepping rotor cipher could be vulnerable to attackers. William Friedman tried to solve this by devising a system that can truly randomize the stepping of the rotors, the resulting is a limited production: M-134. Friedman's associate, Frank Rowlett, tried to solve this as well by constructing each rotor in a way that between one and four output signals were generated, advancing one or more of the rotors. Together with Friedman, a series devices called SIGGOO or M-229, also a combination machine M-134-C were built. In early 1937, US Navy cryptographer Joseph Wenger showed their work to Commander Laurance Safford, the later saw the potential of the machine immediately. So Commander Safford together with another Commander, Commander Seiler added a number of features too, resulting in the Electric Code Machine Mark II [24].

2.3.1 Sigaba Cipher Machine Overview

The design of the SIGABA machine was filed for a patent at December 1944, and the patent was not published until January 2001 [22]. Figure 15 shows a Sigaba cipher.

Figure 15: Sigaba cipher [25]

Sigaba is similar to Enigma and Typex in basic theory, they are all rotor-based classic cipher machines. However, Sigaba is much more complex in its internal structure.

2.3.2 Sigaba Cipher Machine Components

As usual, we only introduce cryptographically significant components, so that Sigaba's working principles can be understood. We will introduce the Sigaba cipher rotor, control rotor and index rotor in this paper.

Sigaba uses 15 rotors in total, but no reflector. These 15 rotors are divided into three banks: cipher rotor bank, control rotor bank and index rotor bank. Figure 16 shows Sigaba's three set of rotors with two individual rotors placed outside. The set of rotors in the very front is index rotor bank, followed by control rotor bank, the set of rotors in the back is cipher rotor bank. The bigger individual rotor is a control or

cipher rotor, they are the same rotor as Enigma cipher rotor, Typex cipher rotor or stator. The smaller individual rotor is an index rotor, it has 10 electronic contacts, which are numbers range from 0 to 9, on each side. Figure 16 shows Sigaba rotors.

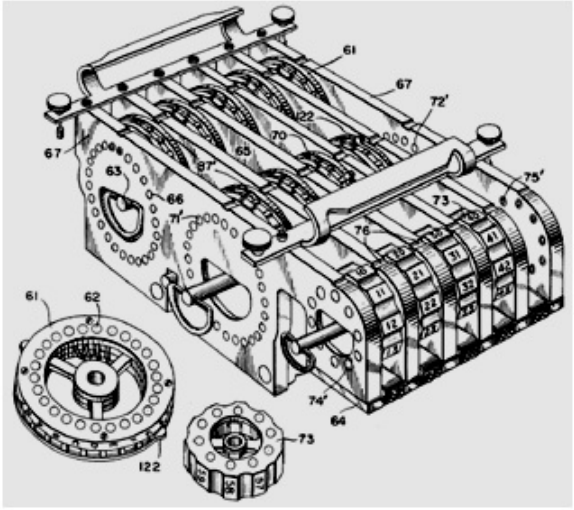

Figure 16: Sigaba rotors [27]

Sigaba Cipher Rotor and Control Rotor

The Sigaba control rotor and cipher rotor are identical and interchangeable since they are the same rotors. In this project, we assume there are 10 alternatives for cipher and control rotors, because of the limitation that a real Sigaba had. Table 4 shows all 10 alternatives.

Like Typex cipher rotor, Sigaba control and cipher can also be placed and step in both forward and reverse directions. If a control or cipher rotor is inserted in forward orientation, it steps anti-alphabetically, like from O to N, N to M as shown in Figure 17.

If a control or cipher rotor is inserted in backward or reverse orientation, it steps alphabetically, like from O to P, P to Q, with letters appearing upside down on the rotors, as shown in Figure 18 [10].

Rotor	Permutation
Rotor 1	YCHLQSUGBDIXNZKERPVJTAWFOM
Rotor 2	INPXBWETGUYSAOCHVLDMQKZJFR
Rotor 3	WNDRIOZPTAXHFJYQBMSVEKUCGL
Rotor 4	TZGHOBKRVUXLQDMPNFWCJYEIAS
Rotor 5	YWTAHRQJVLCEXUNGBIPZMSDFOK
Rotor 6	QSLRBTEKOGAICFWYVMHJNXZUDP
Rotor 7	CHJDQIGNBSAKVTUOXFWLEPRMZY
Rotor 8	CDFAJXTIMNBEQHSUGRYLWZKVPO
Rotor 9	XHFESZDNRBCGKQIJLTVMUOYAPW
Rotor 10	EZJQXMOGYTCSFRIUPVNADLHWBK

Table 4: Ten Sigaba cipher or control rotor alternatives [3]

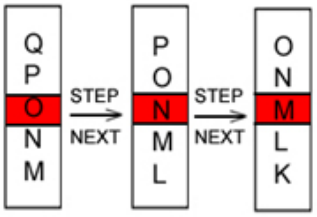

Figure 17: Sigaba rotor in forward orientation [10]

Sigaba Index Rotor

Originally, index rotors can also be inserted backwards, but after June 1945, the operation instruction disallowed the reverse position for index rotors [3]. In this project, we assume that the index rotors are only inserted in forward direction. However unlike Sigaba control or cipher rotor, even index rotor is inserted in forward

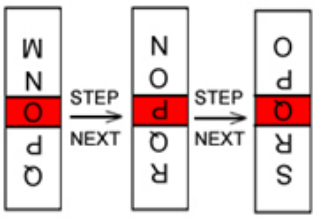

Figure 18: Sigaba rotor in reverse orientation [10]

orientation, numbers appear in a increasing order like shown in Figure 19. The Sigaba index rotor does not step during encrypt or decrypt.

Figure 19: Index rotor orientation [10]

In this project, we assume there are five alternatives for index rotors, because of the limitation that a real Sigaba had. Table 5 shows all five alternatives.

Rotor	Permutation
Rotor 1	7591482630
Rotor 2	3810592764
Rotor 3	4086153297
Rotor 4	3980526174
Rotor 5	6497135280

Table 5: Five Sigaba index rotor alternatives [3]

2.3.3 Sigaba Encryption

When a Sigaba user chooses encryption mode and presses a key at the keyboard to send a letter to encrypt, an electronic signal is generated and is sent to Sigaba. This electronic signal is then separated into two. We name them as Signal I and Signal II here. Signal I goes to the right side of the control rotor bank, actives the four inputs for control rotor bank, which are always 'F','G','H' and 'I', permutes them through the five control rotors following a right to left direction, and emerges them at the left side of control rotor bank. The four outputs of the control rotor bank then

goes through an OR operation to generate the input(s) for the index rotor bank. You can find the OR operation details at Table 6 [3].

Output(s) of control rotors	The n^{th} input of index rotors($n = 1, \ldots, 9$)
B	I_1
C	I_2
D || E	I_3
F || G || H	I_4
I || J || K	I_5
L || M || N || O	I_6
P || Q || R || S || T	I_7
U || V || W || X || Y || Z	I_8
A	I_9

Table 6: Index rotor inputs activation [3]

There are at least one, at most four inputs generated for index rotor bank after this OR operation. These index rotor bank inputs are continually carried by Signal I and go through the index rotor bank following a left to right direction. The outputs of the index rotor bank go through the second OR operation, the result determines which cipher rotor(s) need to step. There are at least one, at most four cipher rotors will step. You can find the second OR operation details at Table 7 [3].

Output(s) of index rotors	The n^{th} cipher rotor ($n = 1, \ldots, 9$)
O_0 || O_9	C_0
O_7 || O_8	C_1
O_5 || O_6	C_2
O_3 || O_4	C_3
O_1 || O_2	C_4

Table 7: Cipher rotor stepping table [3]

From previous analysis about Enigma, we know that a very important security improvement of Sigaba is that its cipher rotors step irregularly. The above series

of actions of control rotors and index rotors that we just explained is how Sigaba performs cipher rotors irregular stepping during encrypt operation, and it is true for decrypt operation as well.

When Signal I goes through control rotor bank, The rightmost and leftmost control rotors in the control rotor bank do not step. But for the three control rotors in the middle, at least one of them, and at most three will step. If there is only one control rotor steps, then it should be the fast one; if there are two control rotors step, it should be the fast one and the middle one; if three control rotors step, then the fast one, the middle one, and the slow one step. Figure 20 shows the positions of the fast, the middle, the slow control rotors and the non-stepping rotors at both sides. We use the following notation in Figure 20 [3]:

F = fast control rotor

M = middle control rotor

S = slow control rotor

X = non-stepping control rotor.

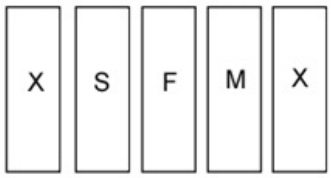

Figure 20: Control rotor bank

Figure 21 shows Signal I journey: it travels through control rotor bank and index rotor bank to determine the stepping of cipher rotors.

Recall that with Sigaba, there are two signals are generated once a key is pressed. We discussed the journey of Sigaba I, now let's take a look at Signal II, which is a

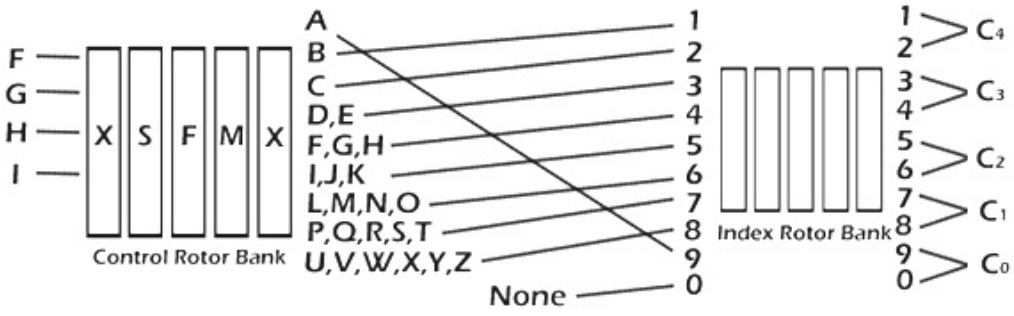

Figure 21: Control and index permutations [26]

relatively straightforward procedure. Signal II goes to the left side of the cipher rotor bank, it then goes through the five cipher rotors in a left to right direction and generates the encrypted letter. Signal I and Signal II journeys are illustrated together at Figure 22.

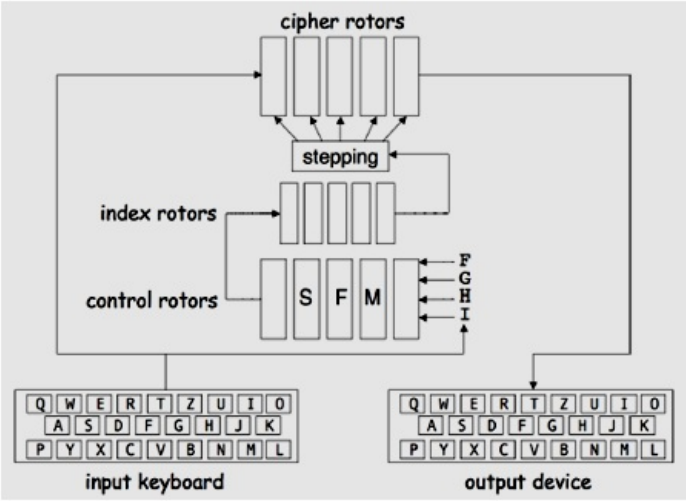

Figure 22: Sigaba encrypt diagram [26]

One more thing that needs to be mentioned is that with Sigaba, the input letter Z is changed to X, and the input space key is changed to Z prior to enciphering [10].

At key setup phase, there are three steps needed before using Sigaba to do encryption:

a. Set up five Sigaba cipher rotors and five control rotors

- Choose five cipher rotors and five control rotors out of 10 alternatives
- Decide the order of 10 rotors
- Set starting position for 10 rotors
- Set orientations of 10 rotors

b. Set up five Sigaba index rotors

- Choose five index rotors out of five alternatives
- Decide the order of five index rotors
- Set starting positions for five index rotors

c. Select encryption mode

2.3.4 Sigaba Decryption

The Sigaba decrypt operation is similar to its encrypt operation, with two differences. First of all, when a Sigaba user chooses decryption mode, and presses a key at the keyboard to decrypt, an electronic signal goes to the right side of the cipher rotor bank, and the decrypted output comes out of the left side of cipher rotor bank. The signal's journey is illustrated at Figure 23.

Secondly, when deciphering, the output letter Z is changed back to the space key; but output letter X is not changed back to Z, since both plaintext X and Z are decrypted as X [10].

At key setup phase, there are three steps are needed before using Sigaba to do decryption: a. Set up five Sigaba cipher rotors and five control rotors

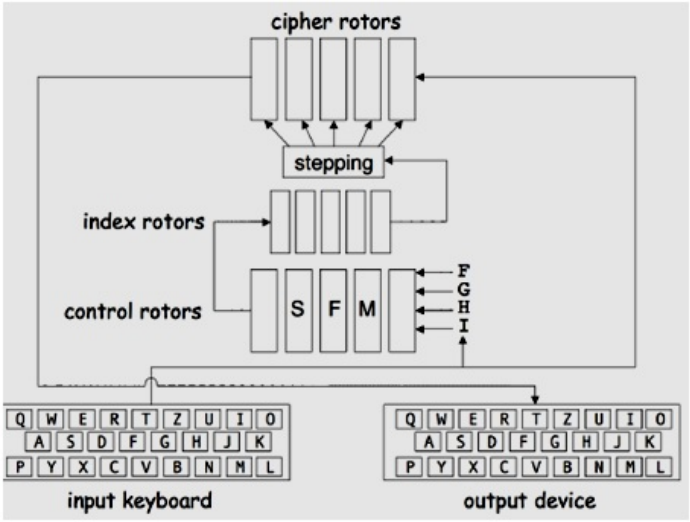

Figure 23: Sigaba decrypt diagram [26]

- Choose five cipher rotors and five cipher rotors out of ten alternatives
- Decide the order of ten rotors
- Set starting position for ten rotors
- Set orientations of ten rotors

b. Set up five Sigaba index rotors

- Choose five index rotors out of five alternatives
- Decide the order of five index rotors
- Set starting position for index rotors

c. Select decryption mode

CHAPTER 3

Simulators

The goal of this project is implementing an application that is used as visual simulators for three classic ciphers used during WWII. It is a web-based software implemented using HTML, CSS, JavaScript and JAVA, etc. This application has been tested on Google Chrome browser, version 18.0.1025.162. Figure 24 is the entrance page of this application.

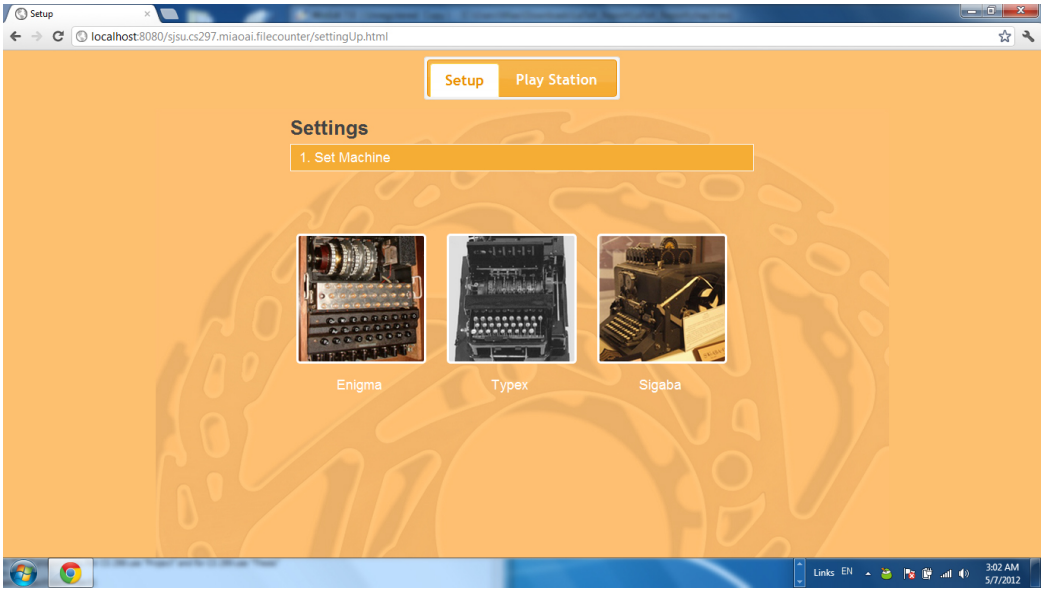

Figure 24: Visual simulator entrance page

From here, users can choose which cipher machine to use. The application entrance page and all steps belong to key setup phase are under a "Setup" tab; the visual simulations for actual operations for each cipher are under a "Play Station" tab.

After choosing a desired cipher machine, a "Next" button shows up at right hand side indicating that the user can go to the next step. Figure 25 illustrates an example

that user chooses Enigma.

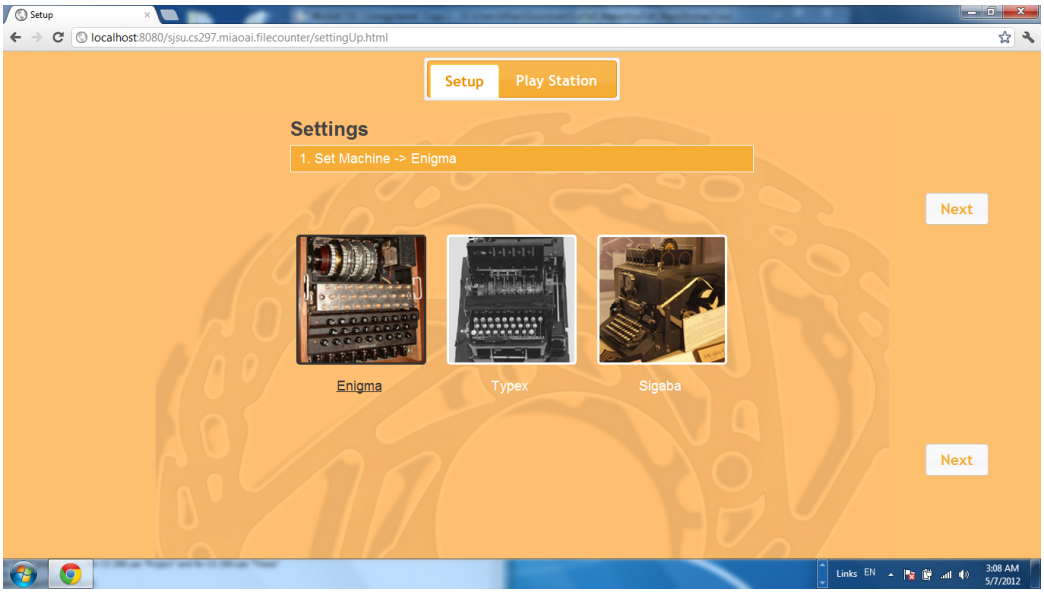

Figure 25: Enigma selected

We introduce the rest of this application in the following order: introducing visualization for various components of all three ciphers first, followed by visualizations for key setup phase of each cipher, the visualization for the actual operations of each cipher is introduced at the end of this chapter.

3.1 Components Visualization

In order to visualize each cipher machine, we need to visualize their components first. From the analysis about three ciphers in the previous chapter, we know that some components, like cipher rotors, are used by all three ciphers; others, like stecker, is unique to one cipher. We will introduce in detail all common components and the unique components.

3.1.1 Keyboard Visualization

A keyboard is used by Enigma, Typex and Sigaba, all three ciphers. Therefore in this project, we visualize keyboard for the Enigma as shown in Figure 26.

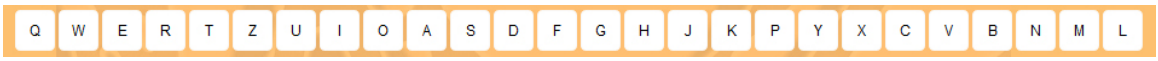

Figure 26: Enigma keyboard visualization

We also visualize keyboard for Typex or Sigaba as shown in Figure 27.

Figure 27: Typex or Sigaba keyboard visualization

The difference is that the keyboard visualization for the Enigma does not contain the space key, but the keyboard visualization for Typex or Sigaba does. This difference matches the real WWII cipher machines' keyboards. The keys on the keyboard are visualized as rectangles labeled with 26 letters, respectively (space key is a rectangle labeled with "SPACE"). All keys are arranged in a straight line. This arrangement is different than the real pattern used by WWII cipher machines' keyboards, but it makes the electronic signal that goes through keys clearly visible to the user. The keyboard visualization appears at the bottom of the visualization for each cipher machine.

A key is highlighted in red when a user presses it. For example, as shown in Figure 28, user just presses letter D, so the D key is highlighted in red.

Figure 28: Keyboard visualization, letter D is pressed

Figure 29 gives another example of what happens when the space key is pressed with Typex. Recall from previous discussions, if the space key is pressed, it transforms

to letter X. Therefore both space key and the X key are highlighted in red at Typex visualization.

Figure 29: Keyboard visualization, space key is pressed

3.1.2 Rotor Visualization

In this project, we use two rotors centering at the same point, but with different radiuses to represent two sides of an individual cipher rotor. The visualization is shown in Figure 30. The number in the center of visualization indicates which rotor it is.

Figure 30: An individual rotor visualization

This visualization is used by Enigma, Typex and Sigaba cipher visualizations. Typex stators use this visualization as well, even though they do not step during operation; Sigaba control rotors also use it even though they are not used as enciphering or deciphering. This is because Typex stators and Sigaba control rotors are interchangeable with their own cipher rotors.

3.1.3 Reflector Visualization

Reflector has contacts on only one side, so instead of using two rotors centered at same point with different radiuses, we use one rotor to represent the Enigma reflector. An individual reflector visualization is shown in Figure 31.

Figure 31: An individual reflector visualization

This reflector visualization is used to visualize both the Enigma and Typex reflector.

3.1.4 Enigma Stecker and Lightboard Visualization

In this project, the simulation of stecker at key setup phase and during the actual operations are visualized in different ways. At Enigma key setup phase, the stecker is represented by two rows of sockets, the upper one and the lower one; a socket is represented as a circle, the circle is highlighted in orange if the socket is already connected; the stecker wiring is represented by red lines in between two rows. This stecker visualization is different than the pattern used by a real WWII Enigma stecker, but it makes the stecker wiring clearly visible to user. Figure 32 shows a default stecker wiring, which is that all letters are connected to themselves. Details of stecker wiring re-configuration will be introduced at Enigma key setup visualization.

For the visualization of the actual operation, in order to show the incoming signal

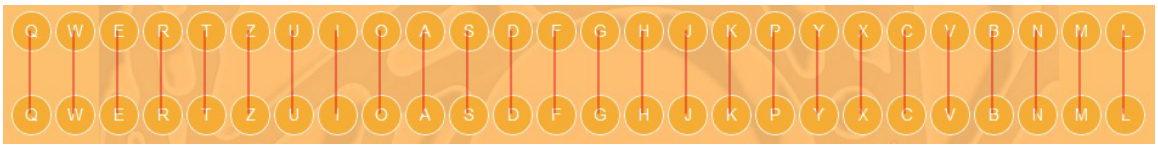

Figure 32: Enigma stecker visualization, default setting

from the keyboard through the stecker, the stecker visualization is placed on top of the keyboard visualization; a red line from the keyboard to the stecker is used to represent this incoming signal when a key is pressed.

We also add a visualization for lightboard on the top of the stecker to simulate how a real WWII Enigma displays its output. Lightboard is a board with 26 bubbles, each bubble corresponds to a letter, a bubble is lighted up if the output of the operation is its corresponding letter. At previous chapter, we do not introduce Enigma lightboard because it is not related to the security of Enigma. The lightboard visualization is semitransparent, in order to simulate the unlit status of the lightboard; each bubble is represented as a circle, the circle is highlighted in opaque yellow if the bubble is lit up; a black line from the stecker to the lightboard is used to represent the outgoing signal when an output is generated. This lightboard visualization is different than the pattern used by a real WWII Enigma lightboard, but it makes outgoing signal clearly visible to user.

Up until now, for the visualization of the actual Enigma operation, we have three rows: the bottom row is the keyboard; the middle row is the stecker; the top row is the lightboard. As illustrated in Figure 33.

Figure 33: Enigma keyboard, stecker and lightboard visualization

You may notice that instead of using two rows of sockets to represent the stecker (like what we do for stecker visualization at Enigma key setup phase), only one row, the middle row, is used to represent stecker. This modification is because that now the stecker wiring can be illustrated by the incoming signal from the keyboard through the stecker; and the outgoing signal from the stecker to the lightboard, so there is no need for two rows of sockets.

3.1.5 Sigaba Index Rotor Visualization

As we already know from previous introduction, Sigaba index rotor has a different structure than its cipher or control rotors. Sigaba index rotor only has 10 numerical contacts on each side. Figure 34 shows the visualization for Sigaba index rotor.

Figure 34: Sigaba index rotor visualization

3.2 Key Setup Visualization

Key setup phases for Enigma, Typex, and Sigaba are visualized separately because each cipher has different setup steps during this phase. We introduce visualizations for Enigma key setup phase first, followed by Typex and Sigaba.

3.2.1 Enigma key Setup Phase Visualization

Visualization for Enigma key setup phase including visualization for Enigma cipher rotors, reflector and stecker setup. We will introduced these one by one.

Visualization for Enigma Cipher Rotors Setup

During the actual Enigma key setup phase, the user needs to choose three cipher rotors, set order, and starting positions for the chosen ones. In this project, this procedure has been visualized as shown in Figure 35 and Figure 36.

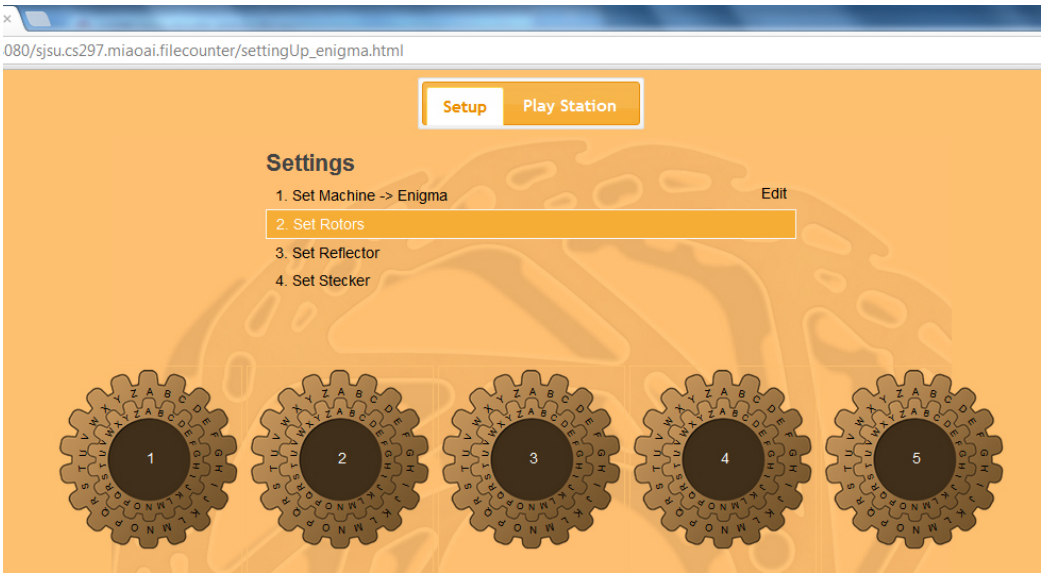

Figure 35: Enigma cipher rotors setup visualization

As it can be seen from Figure 35, there are five alternative rotors, users can choose three out of these five. The order of three chosen rotors is decided by the order of clicking. The first clicked one becomes the leftmost cipher rotor, the second becomes the middle rotor, and the last becomes the rightmost rotor. Meanwhile, an input text box shows up, the user needs to type in a letter to indicate the starting position for each rotor. For example, user clicks Rotor 2 first, so it becomes leftmost cipher rotor, and types in "A" to indicate its starting position; followed by clicking Rotor 1, which becomes middle cipher rotor, and types in "Z" to indicate its starting position; at last, clicking Rotor 3, it becomes rightmost cipher rotor, and types in "T" to indicate its starting position. This procedure is shown in Figure 36.

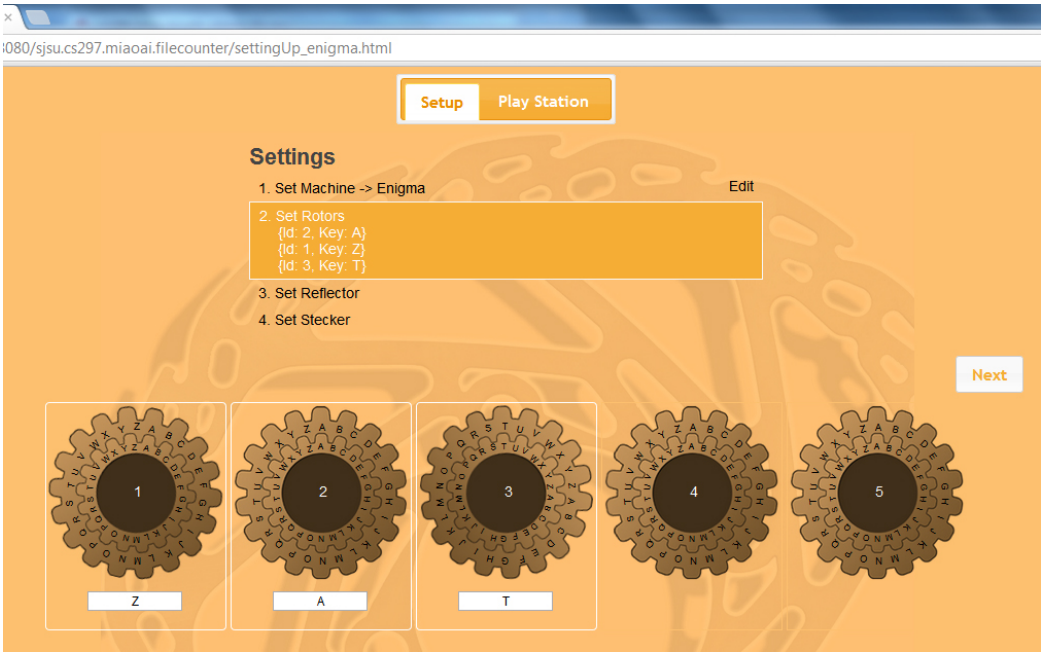

Figure 36: Enigma cipher rotors setup visualization, after setup

From Figure 36, we notice that after the user chooses Enigma cipher rotors, sets order, and starting position, our application records this information as the first part of a Enigma key, and displays them at top of the page. The appearance of "Next" button at the right hand side indicates that user is ready to go to the next step: set up Enigma reflector.

Visualization for Enigma Reflector Setup

After the visualized Enigma cipher rotors setup, we then need to visualize Enigma's reflector setup. In this project, this procedure has been visualized as shown in Figure 37 and Figure 38.

As can be seen from Figure 37, there are two alternative reflectors provided by our application, user can choose the desired one by clicking it. For example, a user chooses Reflector 1 as illustrated in Figure 38.

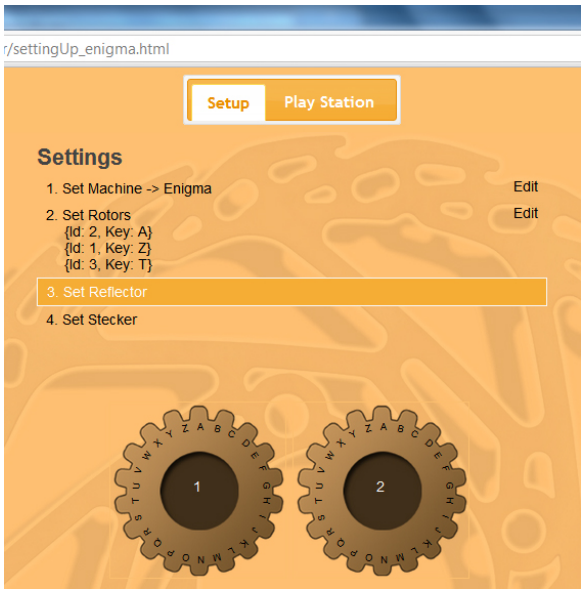

Figure 37: Enigma reflector setup visualization

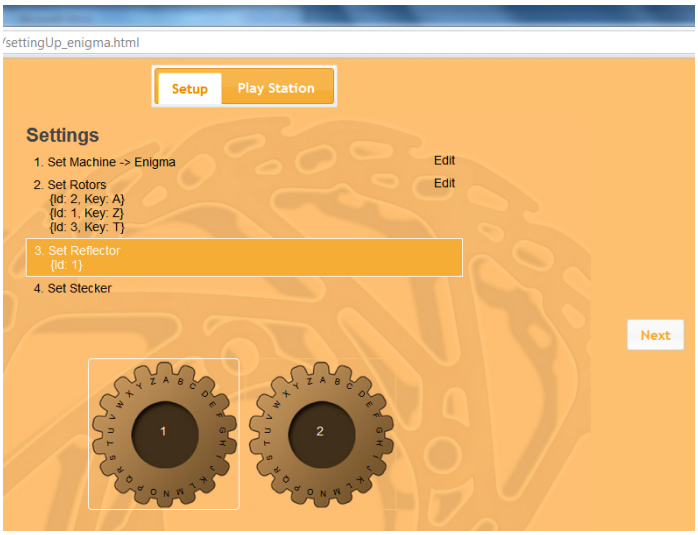

Figure 38: Enigma reflector setup visualization, select Reflector 1

From Figure 38, we notice that after the user finishes clicking Reflector 1, our application records the chosen reflector id as the second part of the Enigma key and displays it at top of the page. Like the Enigma cipher rotor setup, a “Next” button appears at right hand side indicating that the user is ready to go to the next step: set up Enigma stecker.

Visualization for Enigma Stecker Setup

After the visualized Enigma reflector setup, we then need to visualize Enigma stecker setup. In this project, this procedure starts from Figure 39.

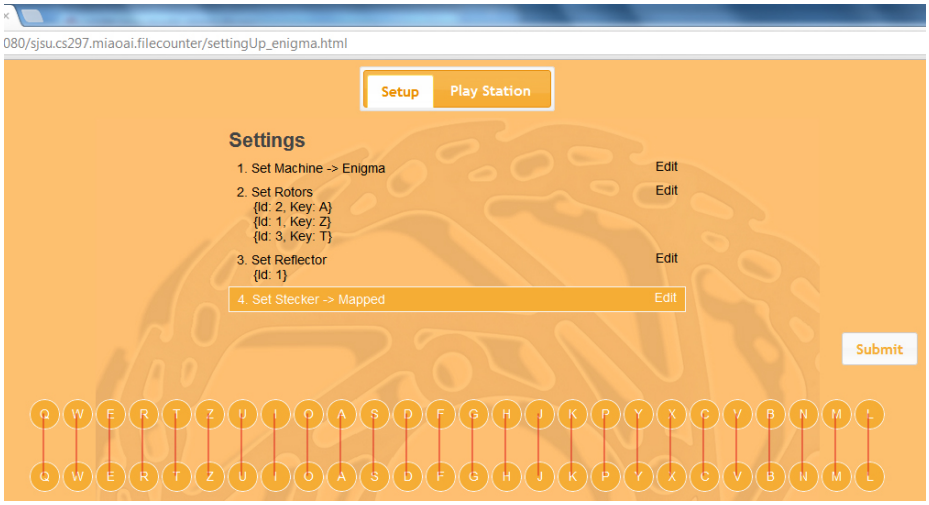

Figure 39: Enigma stecker setup visualization, default setting

Our application provides a default stecker setting as shown in Figure 39 at first, that means all letters are connected to themselves. User can re-configure this setting by de-selecting existing pairs and re-wiring letters into different pairs. For example, the user wants to re-wire letter A to letter J, and re-wire letter S to letter O, he/she needs to single click the letter A and letter F on the bottom to de-select default pair (A-A), and (J-J); single click the letter S and letter O on the bottom to de-select default pair (S-S), and (O-O). As illustrated in Figure 40.

Now the user can re-wire letter A to letter J by single clicking letter A on the bottom, and single clicking letter J on the top; single clicking letter J on the bottom, and single clicking letter A on the top to re-wire letter A to letter J as a stecker pair. This procedure is illustrated in Figure 41.

The user can continue to re-wire letter S to letter O by single clicking letter S

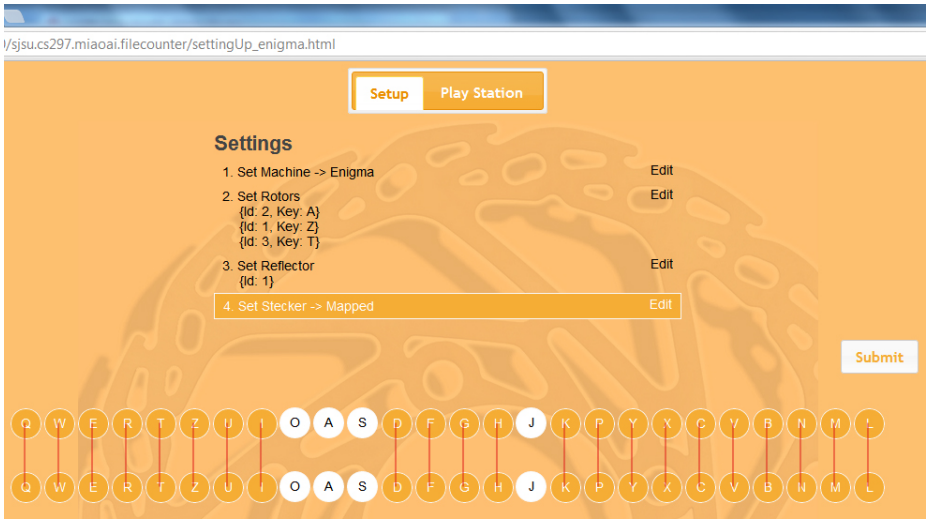

Figure 40: Enigma stecker setup visualization, de-select default setting

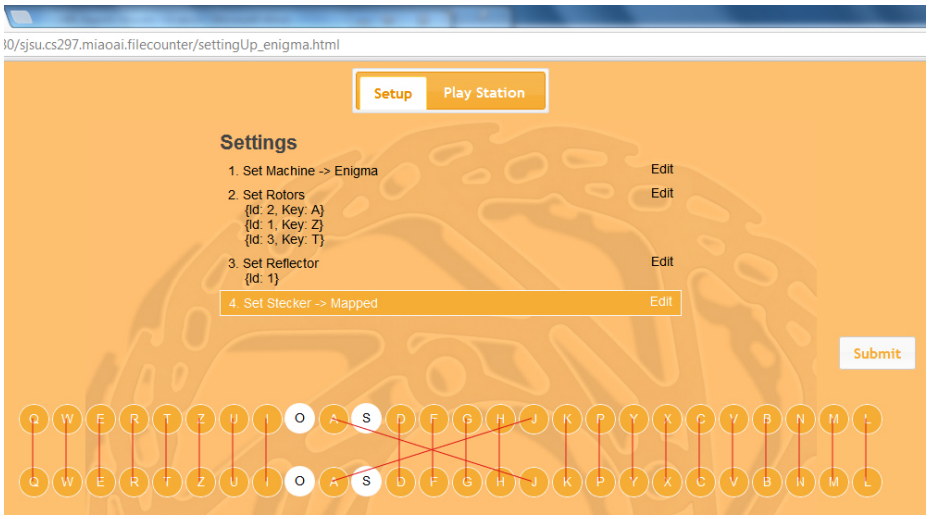

Figure 41: Enigma stecker setup visualization, re-wire A to J

on the bottom, single clicking letter O on the top; and single clicking letter O on the bottom, and single clicking letter S on the top to re-wire letter S to letter O as a stecker pair. This procedure is illustrated in Figure 42.

It can be seen from Figure 42 that our application records three Enigma cipher rotors ids, clicking order, their starting positions, Enigma reflector id, and Enigma stecker setting as an Enigma key. Our application displays this at the top of the page.

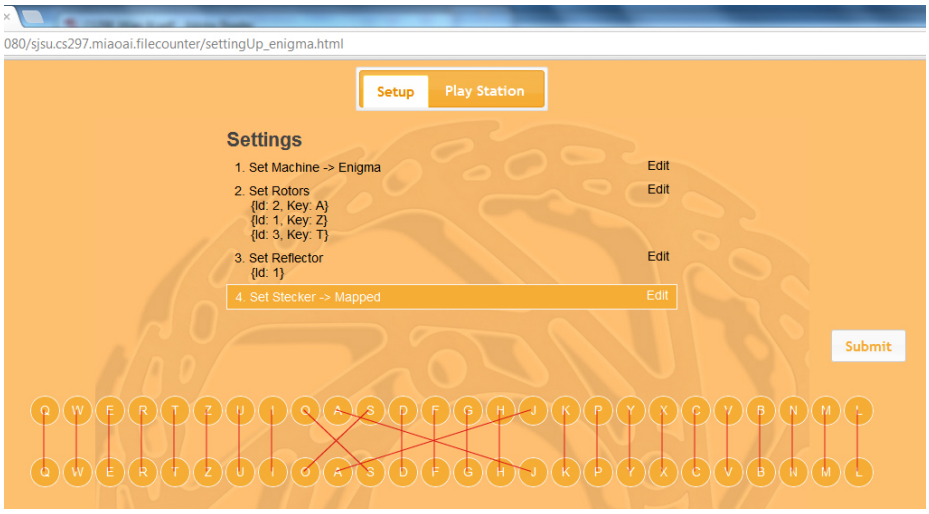

Figure 42: Enigma stecker setup visualization, re-wire S to O

So a complete sample Enigma key from this given example is given in Table 8.

Leftmost cipher rotor: Rotor 2, Starting position: A
Middle cipher rotor: Rotor 1, Starting position: Z
Rightmost cipher rotor: Rotor 3, Starting position: T
Reflector: Reflector 1
B-B, C-C, D-D, E-E, F-F, G-G, H-H, I-I, K-K,L-L, M-M, N-N,
P-P, Q-Q, R-R, T-T, U-U, V-V,W-W, X-X, Y-Y, Z-Z, A-J, O-S

Table 8: A complete sample Enigma key

We will need this sample Enigma key for visual simulation of the Enigma cipher at section 3.3. After the user constructs all stecker pairs, a "Submit" button appears at right hand side as shown at Figure 42. The appearance of this button indicates that the user has finished setting up Enigma key, and is ready to encrypt or decrypt using Enigma.

3.2.2 Typex key Setup Phase Visualization

Visualization for Typex key setup phase focuses on introducing the visualization for Typex cipher rotor and stator setup. Visualization for Typex reflector setup

is omitted since there is no alternatives from what we already know from previous introduction, a reflector is added by our application behind the scenes.

Visualization for Typex Cipher and Stator Setup

As we already know from introduction about Typex cipher, at key setup phase, the user needs to choose three cipher rotors and two stators out of eight alternatives, set order, starting positions and select orientations. In this project, this procedure has been visualized as shown in Figure 43 and Figure 44.

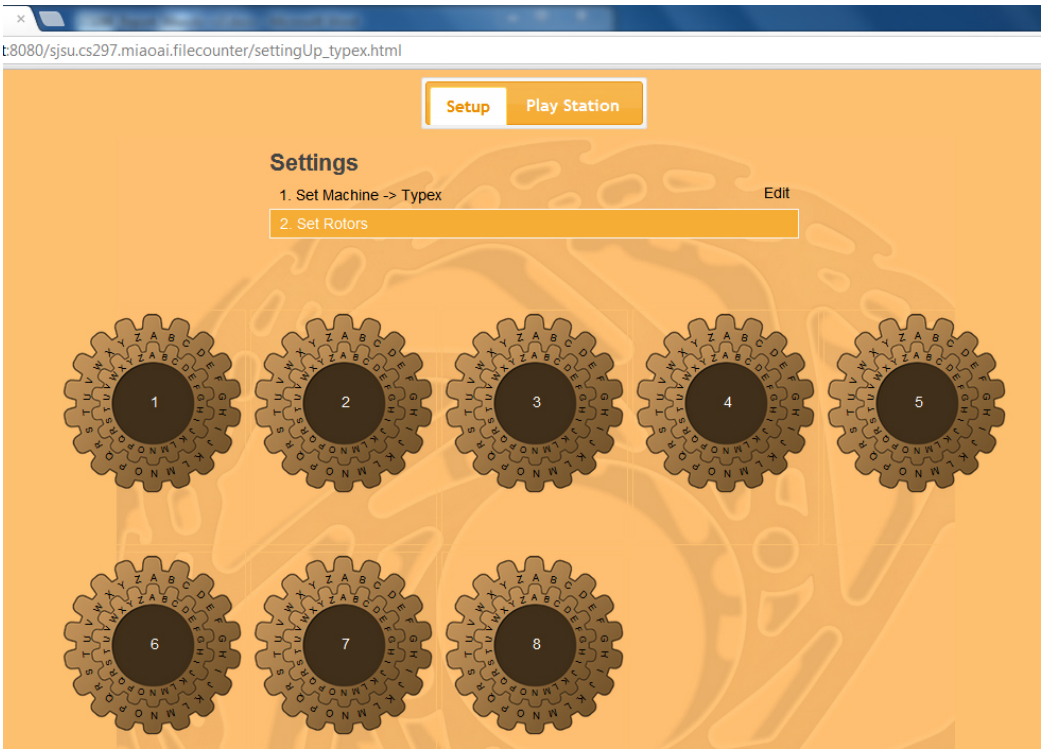

Figure 43: Typex rotors setup visualization

As it can be seen from Figure 43, there are eight alternatives, the user can choose three cipher rotors and two stators out of these eight rotors. Like setting up Enigma cipher rotor, the order of the Typex cipher rotors and stators is decided by the order of clicking; the starting positions for each chosen one is decided by the typed-in letter

in the text box; But for Typex cipher rotors and stators, the user needs to select orientations for each chosen one as well. Orientation for each chosen rotor is decided by the selection from the dropdown box. For example, user selects "Reversed" as orientation for Rotor 1, and Rotor 3; selects "Forward" as orientation for Rotor 2, Rotor 4 and Rotor 5. Figure 44 illustrates this.

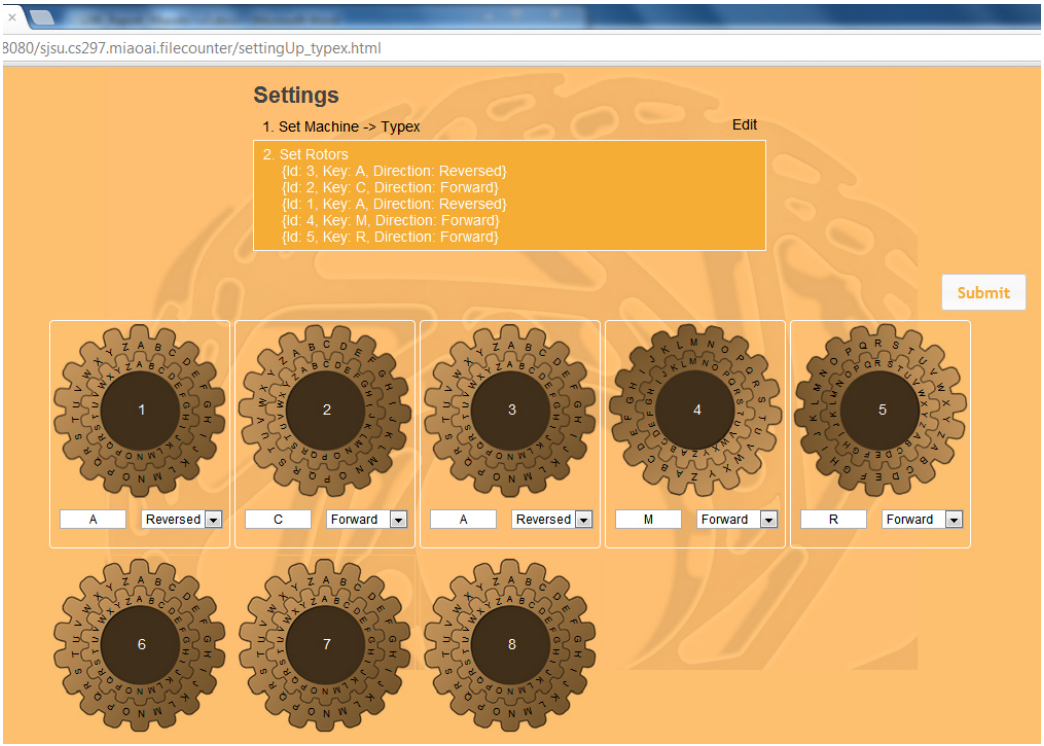

Figure 44: Typex rotors setup visualization, after setup

It can be seen from Figure 44 that our application records three Typex cipher rotors and two Typex stator rotors ids, clicking order, their starting positions, and orientations as Typex key. Our application displays this at top of the page. So a complete Typex sample key from this given example is given in Table 9.

We will need this sample Typex key for the visual simulation of Typex cipher at section 3.3. A "Submit" bottom shows up at the right hand side indicating that the user has finished setting up Typex key, and is ready to do encrypt or decrypt using

Leftmost cipher rotor: Rotor 3, Starting position: A, Orientation: Reversed
Middle cipher rotor: Rotor 2, Starting position: C, Orientation: Forward
Rightmost cipher rotor: Rotor 1, Starting position: A, Orientation: Reversed
Left stator: Rotor 4, Starting position: M, Orientation: Forward
Right stator: Rotor 5, Starting position: R, Orientation: Forward

Table 9: A complete sample Typex key

Typex.

3.2.3 Sigaba key Setup Phase Visualization

Visualization for Sigaba key setup phase includes visualizations for Sigaba cipher rotors, control rotors, Sigaba index rotors and selecting operation mode. We will introduce these one by one.

Visualization for Sigaba Cipher and Control Rotors Setup

As we already know, at Sigaba key setup phase, the user needs to choose five cipher rotors and five control rotors. The user needs to set order, starting position and select orientations for each chosen cipher and control rotor as well. In this project, visualization for the Sigaba cipher rotors setup is shown in Figure 45 and Figure 46.

As it can be seen from Figure 45, there are 10 alternative rotors, user can choose five as cipher rotors and five as control rotors. Like Enigma, the order of these five cipher rotors and five control rotor is decided by the order of clicking. Starting position and orientation for each chosen rotor are decided using the same method as Typex.

It can be seen from Figure 46, our application records Sigaba cipher rotor ids, clicking order, their starting positions, and orientations as the first part of a Sigaba key, and displays it at top of the page. A “Next” button appears at right hand side

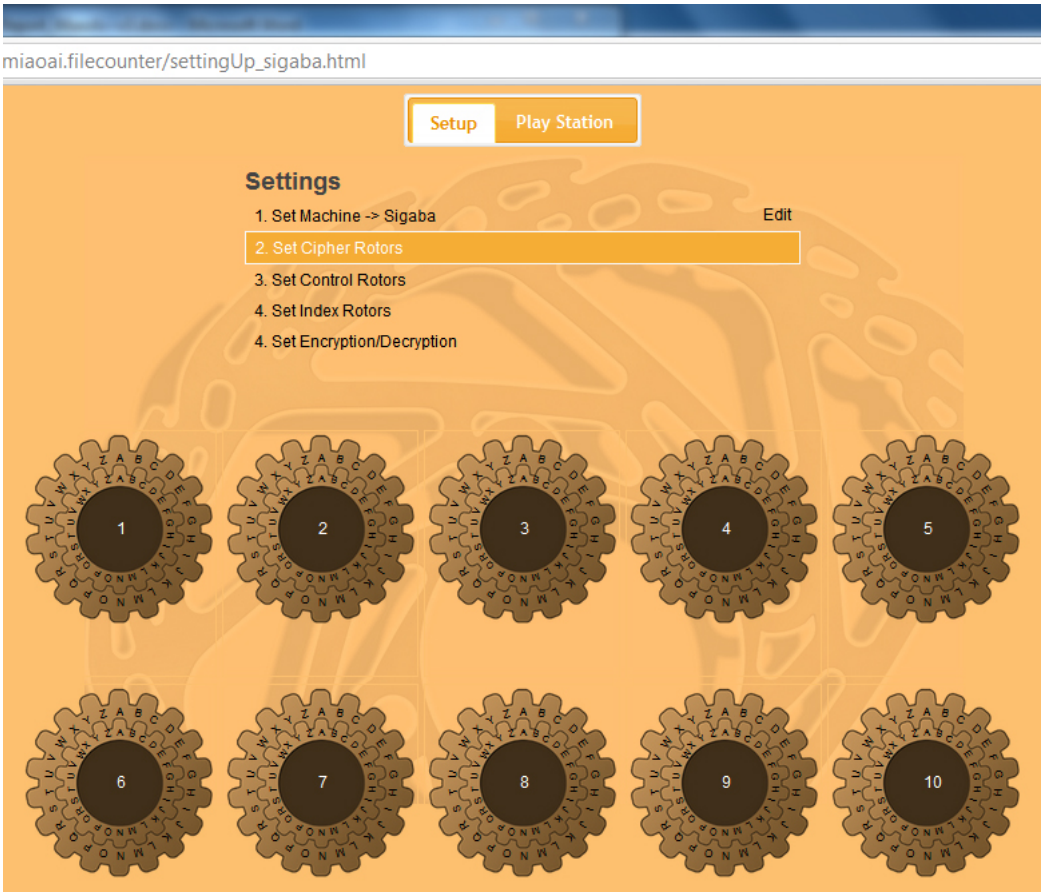

Figure 45: Sigaba rotors setup visualization

indicating that the user is ready to go to the next step: set up Sigaba control rotor.

After chosing Sigaba five cipher rotors, the left five rotors become Sigaba control rotors automatically. Figure 47 and Figure 48 illustrate this procedure.

The order of these five Sigaba control rotors, their starting position and orientation are decided using the same method as Sigaba cipher rotor.

It can be seen from Figure 48, the chosen Sigaba control rotors information, including Sigaba control rotor ids, starting positions, and orientations are recorded by our application and becomes the second part of the Sigaba key. A "Next" button appears at right hand side indicating that the user is ready to go to the next step:

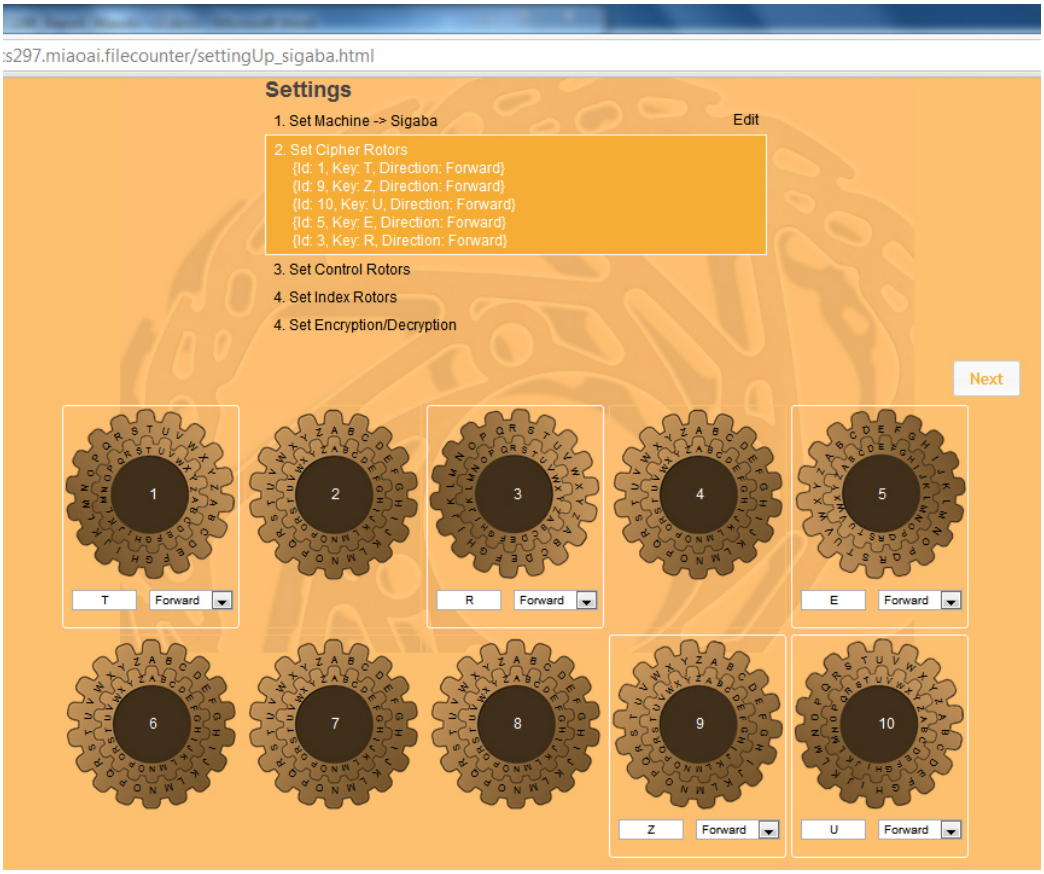

Figure 46: Sigaba cipher rotors setup visualization, after setup

set up Sigaba index rotor.

Visualization for Sigaba Index Rotor Setup

After the visualized Sigaba cipher and control rotors setup, we then need to visualize how to do key setup for the Sigaba index rotor. In this project, this procedure has been visualized as shown in Figure 49 and Figure Figure 50.

As it can be seen from Figure 49, the order of these five Sigaba index rotors, and starting positions for each index rotor are set using the same method as the Sigaba cipher rotor. However, instead of typing in a letter, user types in a number ranges from 0 to 9 to determine the starting position, as shown in Figure 50.

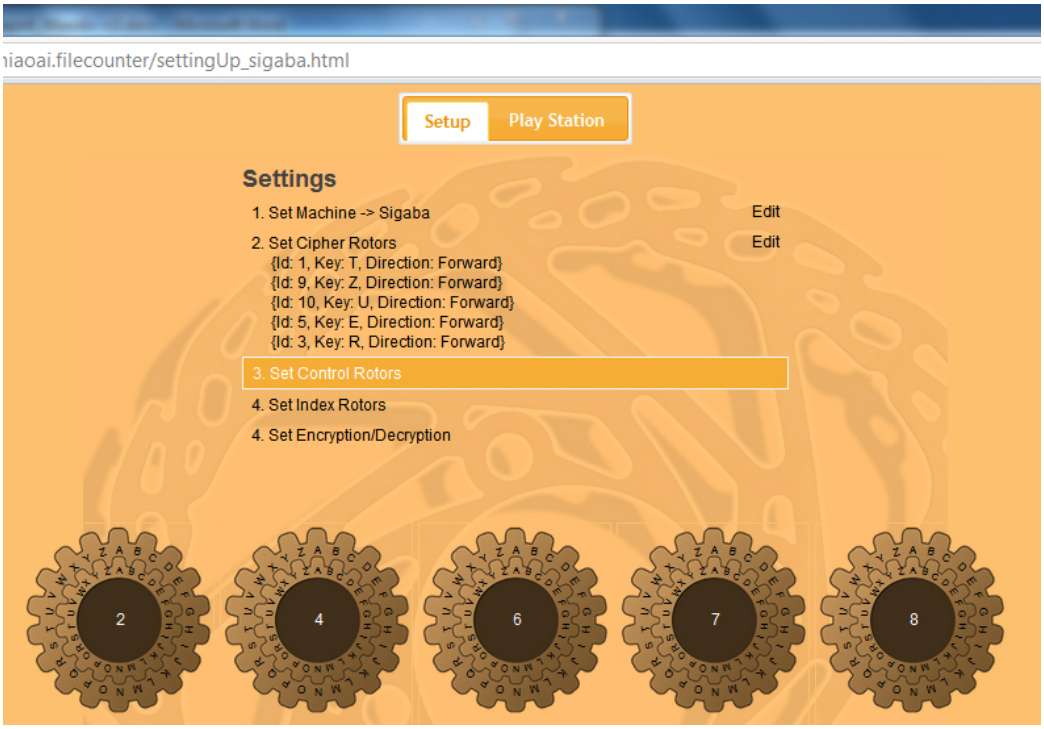

Figure 47: Sigaba control rotors setup visualization

As can be seen from Figure 50, after the user determines the order of index rotors, and the starting position for each index rotor, this information is recorded by our application as the third part of the Sigaba key and is displayed at the top of page as well. A "Next" button appears at right hand side indicating that the user is ready to go to the next step: decide operation mode.

Visualization for Operation Mode Selection

As we already know from previous analysis, unlike Enigma or Typex, Sigaba encrypt and decrypt are slightly different procedures. Therefore the user needs to decide whether he/she wants to do encrypt or decrypt during key setup phase. In this project, we visualize operation selection as the last step as shown in Figure 51 and Figure 52.

As can be seen from the bottom of Figure 51, there are two options available:

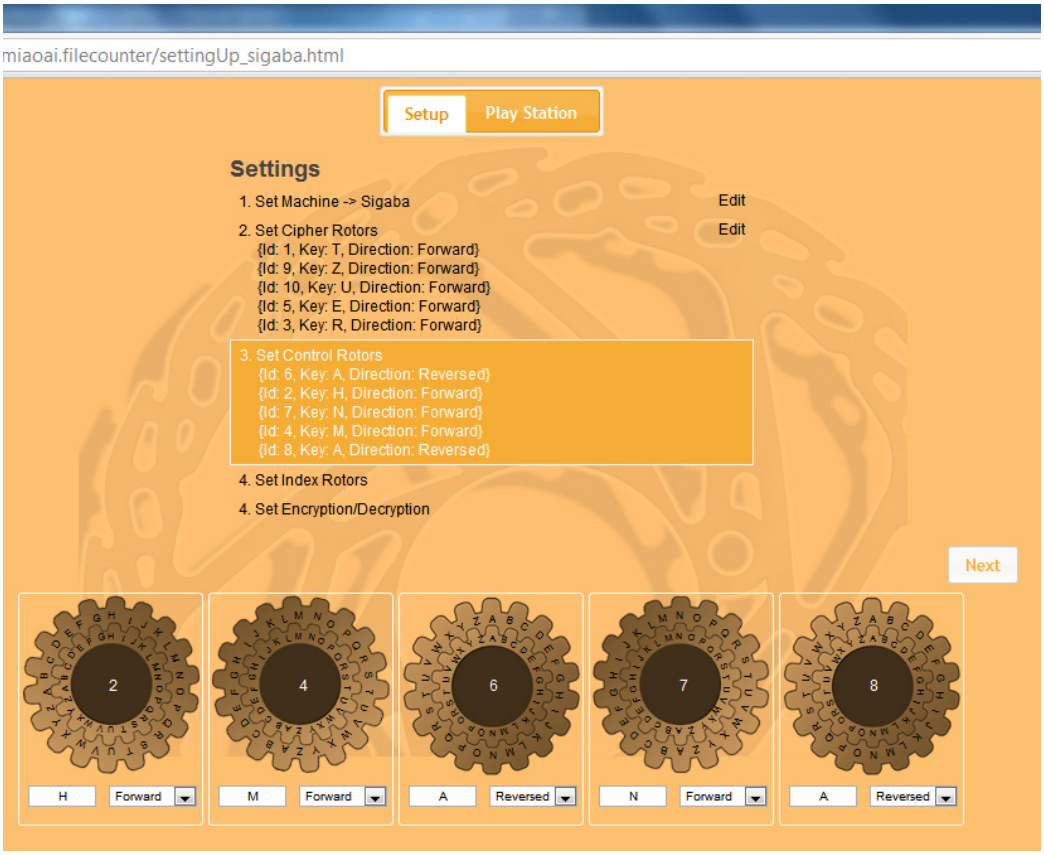

Figure 48: Sigaba control rotors setup visualization, after setup

encryption and decryption. The user can select the desired mode by clicking it. For example, if the user wants to do encrypt using Sigaba, he/she can click the button labeled "Encryption", as illustrated in Figure 52.

After the user finishes clicking "Encryption" button, the user finishes the last step of Sigaba key setup phase. This selection is recorded by our application as the fourth part of the Sigaba key and displays it at top of page as well. So a complete Sigaba sample key from this series of examples is given in Table 10.

We will need this sample Sigaba key for visual simulation of Sigaba cipher at section 3.3. As usual, a "Submit" appears at right hand side indicating that the user has finished setting up Sigaba key , and ready to do operation, in this case,

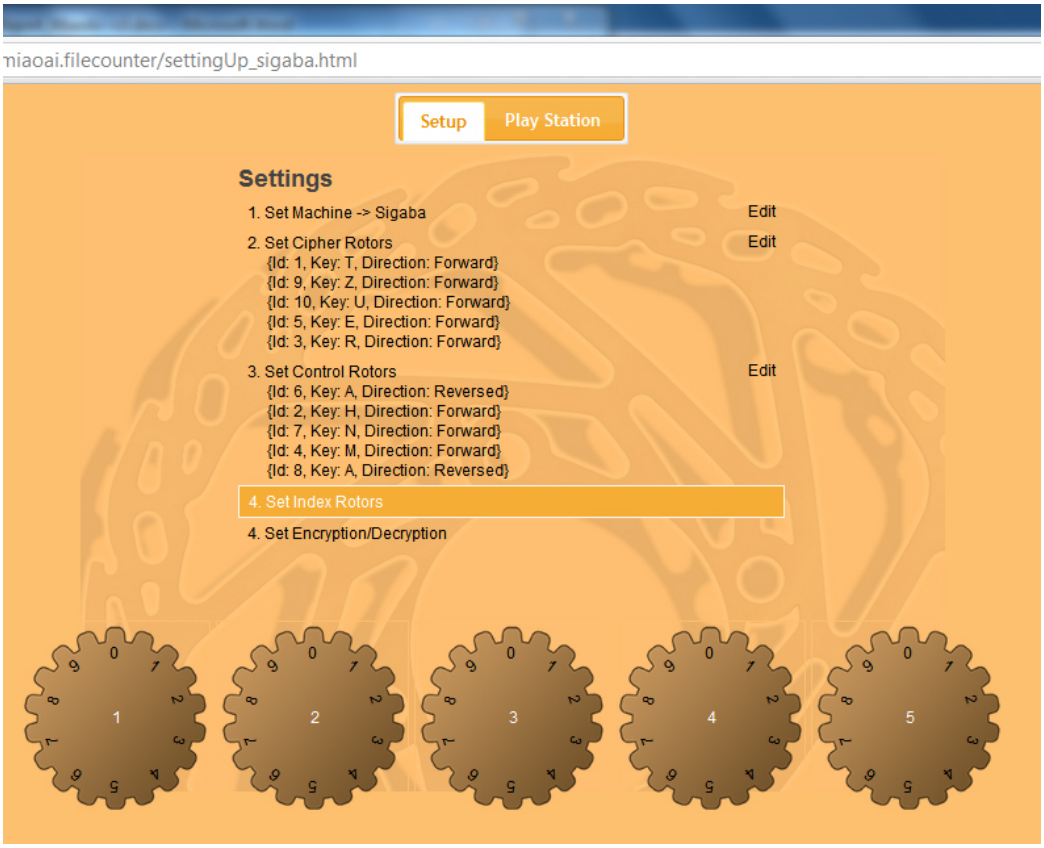

Figure 49: Sigaba index rotors setup visualization

encryption, using Sigaba.

3.3 Visual Simulations for Three Classic Ciphers

In this project, since our goal is to show the internal structures and exhibit signal journey(s) so that the user can understand the working principles, we do not do external structures visualization for any cipher.

By far, we have already visualized all cryptographic important components for Enigma, Typex, and Sigaba, plus key setup phases for them as well. From here, we have everything ready to do visual simulations of the actual operations for three ciphers. As usual, we start from Enigma first, followed by Typex and Sigaba.

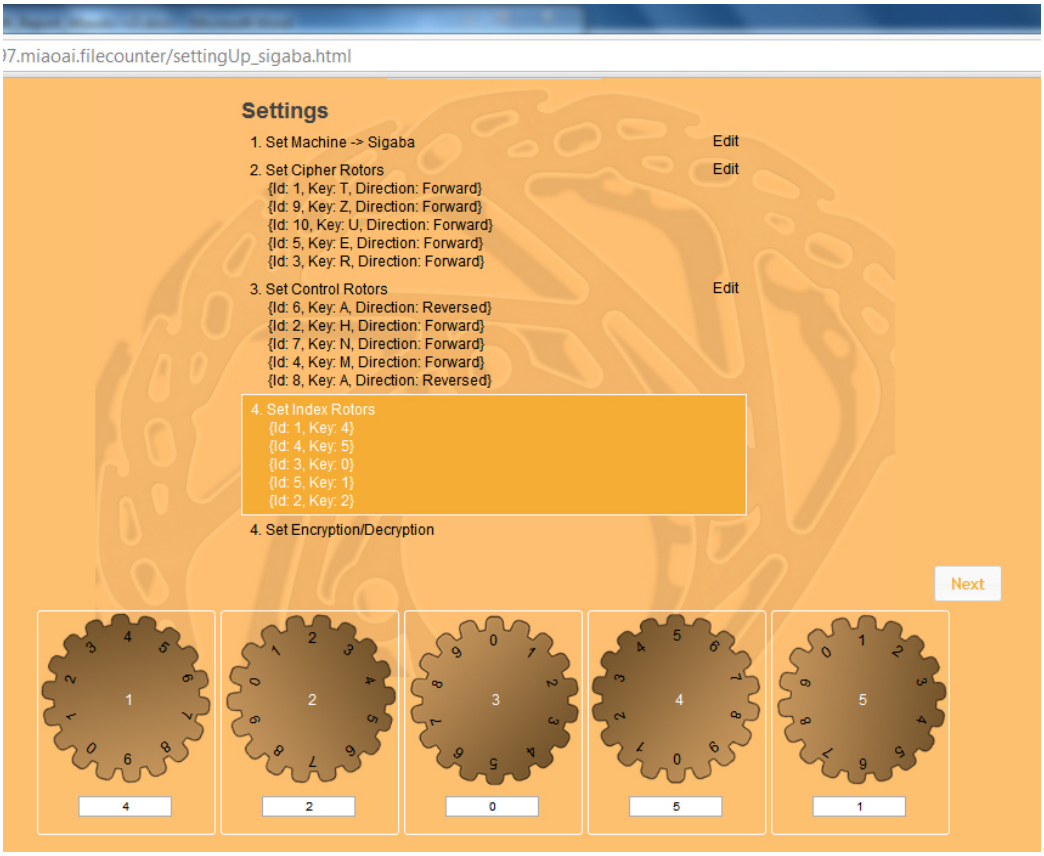

Figure 50: Sigaba index rotors setup visualization, after setup

3.3.1 Visual Simulation for Enigma

When we do visual simulation for the Enigma cipher, we arrange three Enigma cipher rotors at the top of the page, place leftmost cipher rotor at top left, middle cipher rotor at top middle, and rightmost cipher rotor at top right; each rotor is adjusted automatically by our application so that starting letter is at 12 o'clock position. We place the Enigma reflector at the middle left of the page, Enigma keyboard, stecker, and lightboard at the bottom.

Recall the sample Enigma key we get as we introduce the visualization for the Enigma key setup phase, using that sample key, this means arrange Rotor 2 at top left, Rotor 1 at top middle, and Rotor 3 at top right; three cipher rotors are adjusted

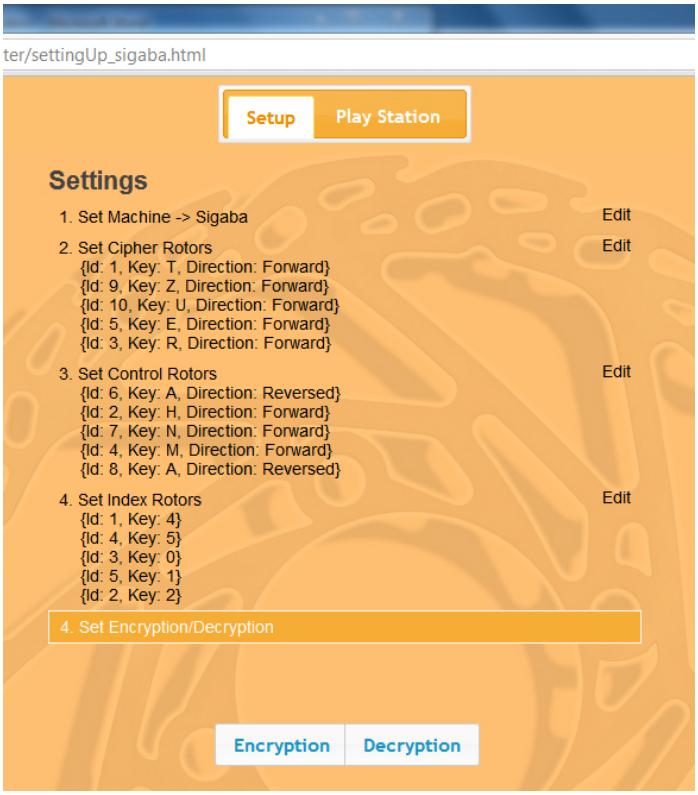

Figure 51: Sigaba operation mode selection visualization

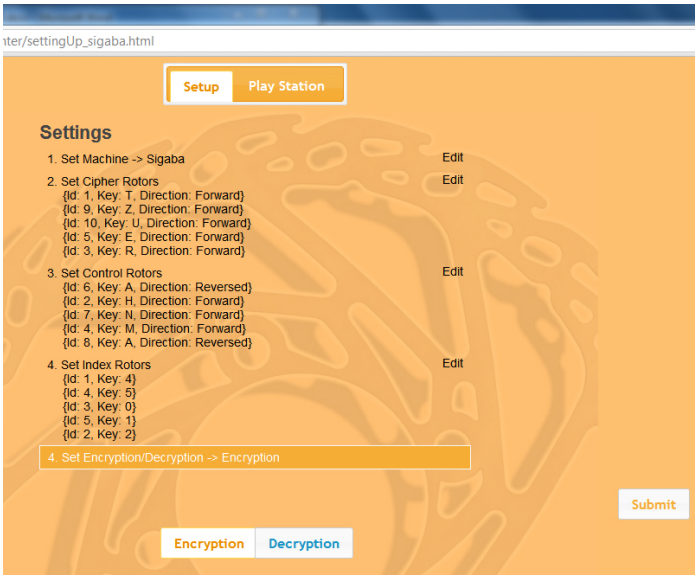

Figure 52: Sigaba operation mode selection visualization, choose encryption

From left to right
cipher rotor: Rotor 1, Starting position: T, Orientation: Forward
cipher rotor: Rotor 9, Starting position: Z, Orientation: Forward
cipher rotor: Rotor 10, Starting position: U, Orientation: Forward
cipher rotor: Rotor 5, Starting position: E, Orientation: Forward
cipher rotor: Rotor 3, Starting position: R, Orientation: Forward
From left to right
control rotor: Rotor 6, Starting position: A, Orientation: Reversed
control rotor: Rotor 2, Starting position: H, Orientation: Forward
control rotor: Rotor 7, Starting position: N, Orientation: Forward
control rotor: Rotor 4, Starting position: M, Orientation: Forward
control rotor: Rotor 8, Starting position: A, Orientation: Reversed
From left to right
index rotor: Rotor 1, Starting position: 4
index rotor: Rotor 4, Starting position: 5
index rotor: Rotor 3, Starting position: 0
index rotor: Rotor 5, Starting position: 1
index rotor: Rotor 2, Starting position: 2
Encryption

Table 10: A complete sample Sigaba key

automatically so that letter A, letter Z and letter T are at 12 o'clock position for each of them; Reflector 1 is placed at middle left of the page. This example is shown in Figure 53.

As we already know, when an encrypt or decrypt operation begins, an electronic signal is activated and goes through stecker, cipher rotors and reflector and goes back in a inverse direction. In this project, we use color lines to represent signals, red line as incoming signal, and black line as outgoing one. The keys on the keyboard, sockets on the stecker and bubbles on the lightboard are highlighted using red, orange and yellow, respectively, when they are pressed by user or touched by signals; all contacts on three cipher rotors and reflectors are circled with white borders when they are touched by signals. For example, when the user types in letter A, it is re-wired to

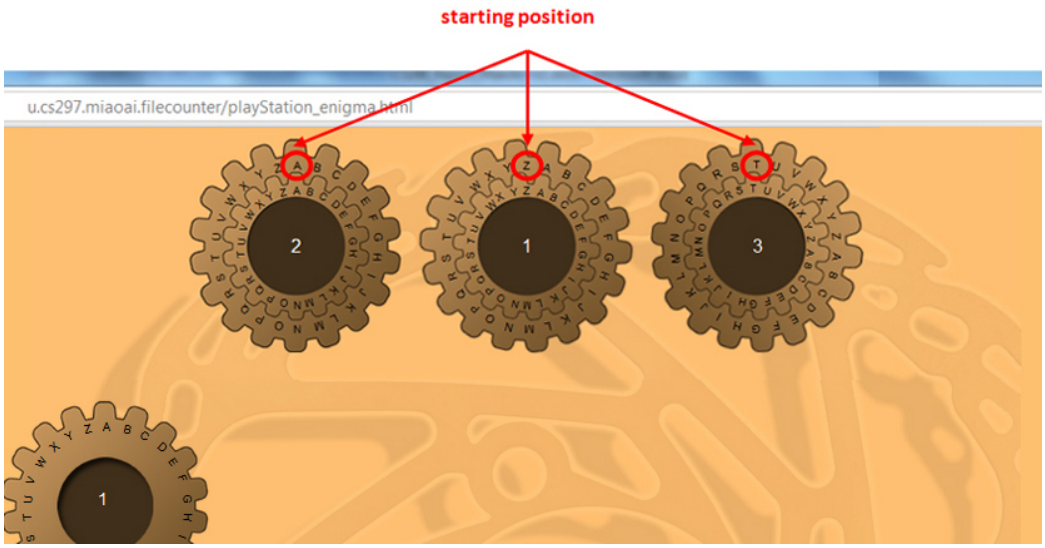

Figure 53: Visual simulation of Enigma

letter J, so key A on the keyboard is in red and socket J on stecker is in orange; incoming signal arrives at Rotor 3, and touches contact D, then contact H, incoming signal goes out of Rotor 3 and arrives at Rotor 1, it touches contact M, then contact O, incoming signal goes out of Rotor 1 and arrives at Rotor 2, it touches contact P, then contact C, incoming signal goes out of Rotor 2 and arrives Reflector 1, touches contact C, then contact U, incoming signal now transforms to outgoing signal, it goes out of Reflector 1 and arrives back at Rotor 2, it touches contact U, then contact H, outgoing signal goes out of Rotor 2 and arrives back at Rotor 1, it touches contact G, then contact F, outgoing signal goes out of Rotor 1 and arrives back at Rotor 3, it touches contact A, then contact T, so all these contacts mentioned are circled with white borders. Outgoing signal goes out of Rotor 3 and arrives back at stecker, touch sockets Z, so sockets Z on stecker is in orange; and finally outgoing signal lights bubble Z up, so bubble Z on the lightboard is in yellow. This signal journey is illustrated at Figure 54.

A button labeled “Input and Output” is at left side of the page. It stores all

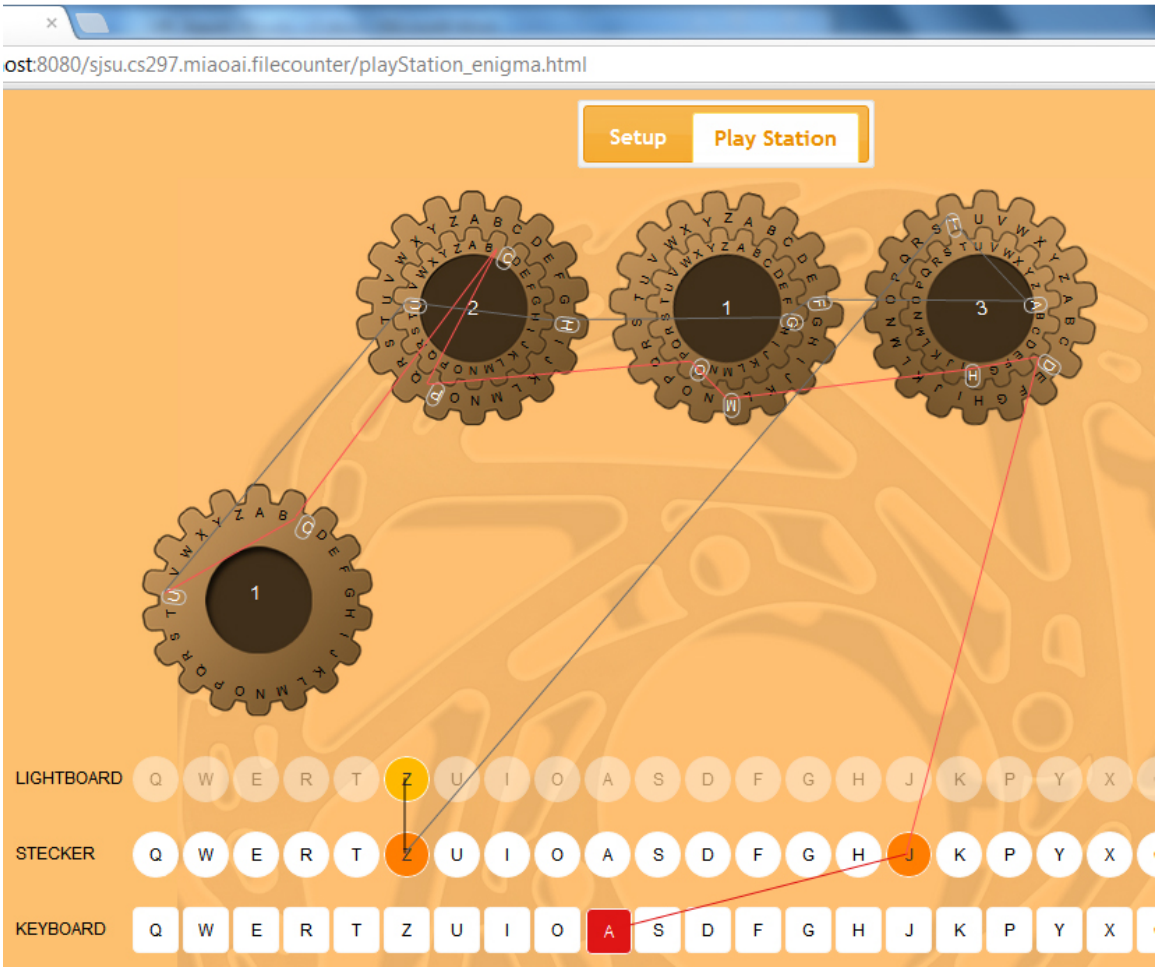

Figure 54: Visual simulation of Enigma, operation begins

inputs and outputs as illustrated in Figure 55. The user can click to open it and see.

For example, we type in a plaintext "AIMIAOCSSJSUPROJECT", and get ciphertext as "ZWBHCUMWLXOQCOCRHPW" as illustrated in Figure 56.

The ciphertext has been compared with the ciphertext generated from a Flash simulator, they are the same [2]. Ciphetext from the Flash simulator is illustrated at Figure 57.

Use ciphertext "ZWBHCUMWLXOQCOCRHPW" as input, with the exactly

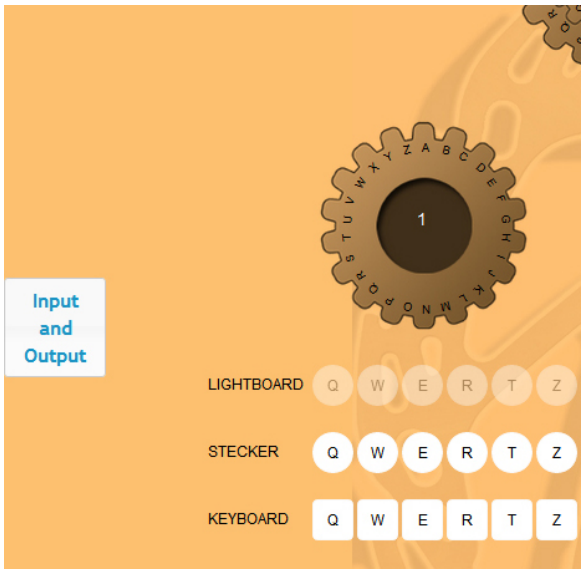

Figure 55: Visual simulation of Enigma, "Input and Output" box

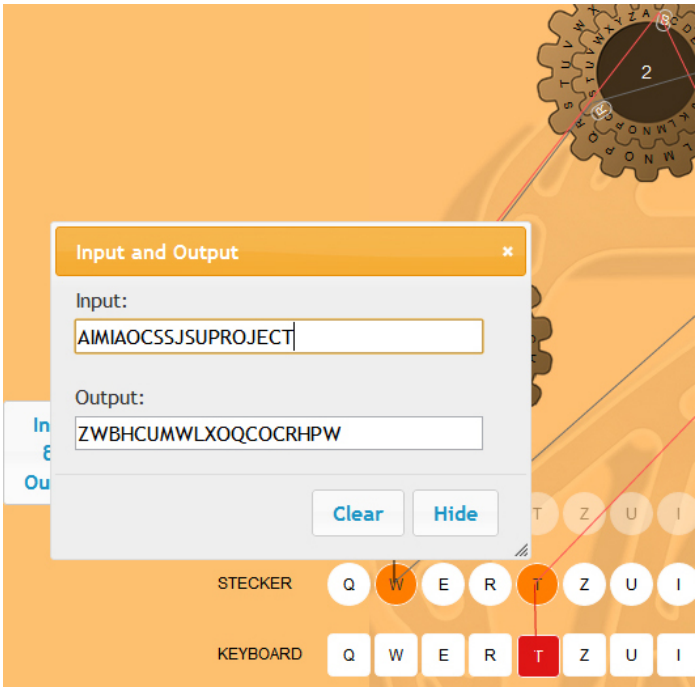

Figure 56: Visual simulation of Enigma, encrypt output

same key setup, we run our Enigma simulator again, and get "AIMIAOCSSJSUPRO-JECT" as the plaintext as shown in Figure 58.

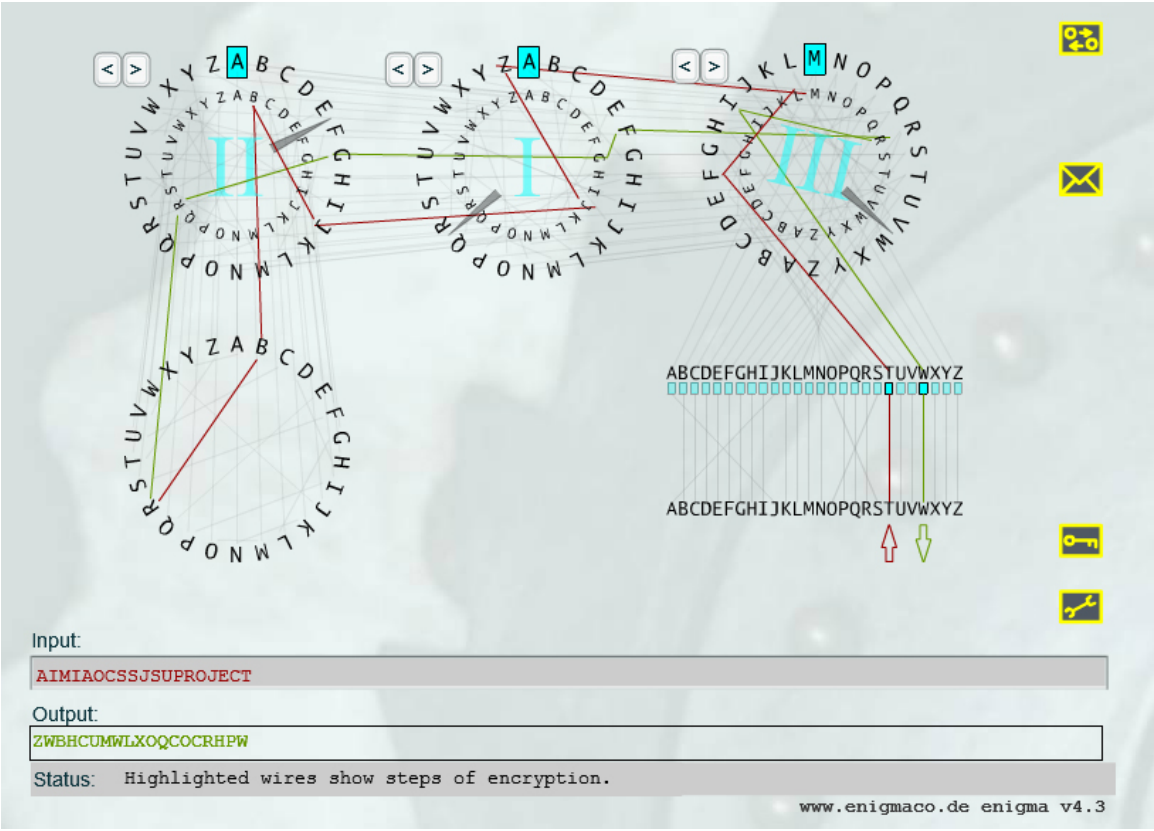

Figure 57: Output of the Enigma Flash simulator

This confirms that our visual simulator can encrypt and decrypt message correctly. More examples and test result can be found at Appendix A.

Recall at Chapter 2, we introduce a special occasion, a "double stepping" for middle rotor stepping, we give an example at here. We type a long message to find a position that when the next key is pressed, the leftmost rotor will be activated by middle rotor to step, as illustrated in Figure 59.

In Figure 59, the middle rotor, Rotor 2 is at position "E", and the leftmost rotor, Rotor 1 is at position "A". As we already know, Rotor 2's notch is also "E", so when the next key is pressed, leftmost rotor, Rotor 1, will be activated and steps from "A" to "B". What we want to verify is that the middle rotor also steps, from "E"

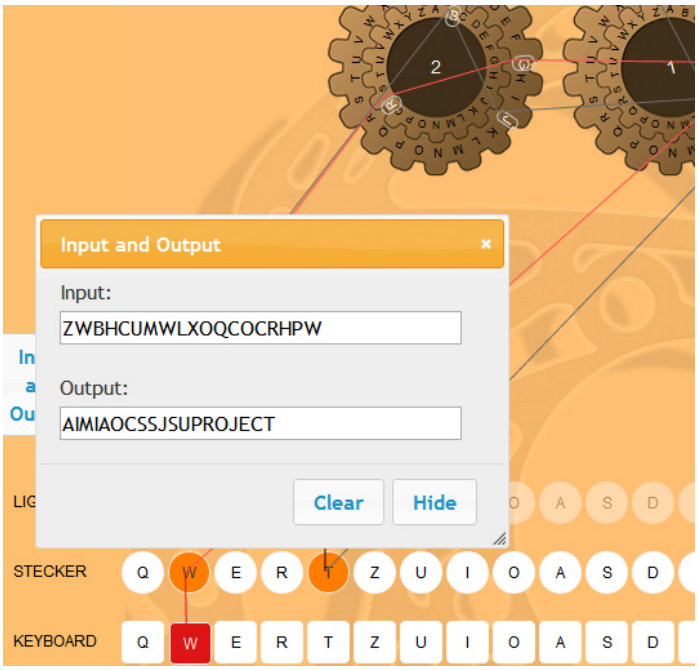

Figure 58: Visual simulation of Enigma, decrypt output

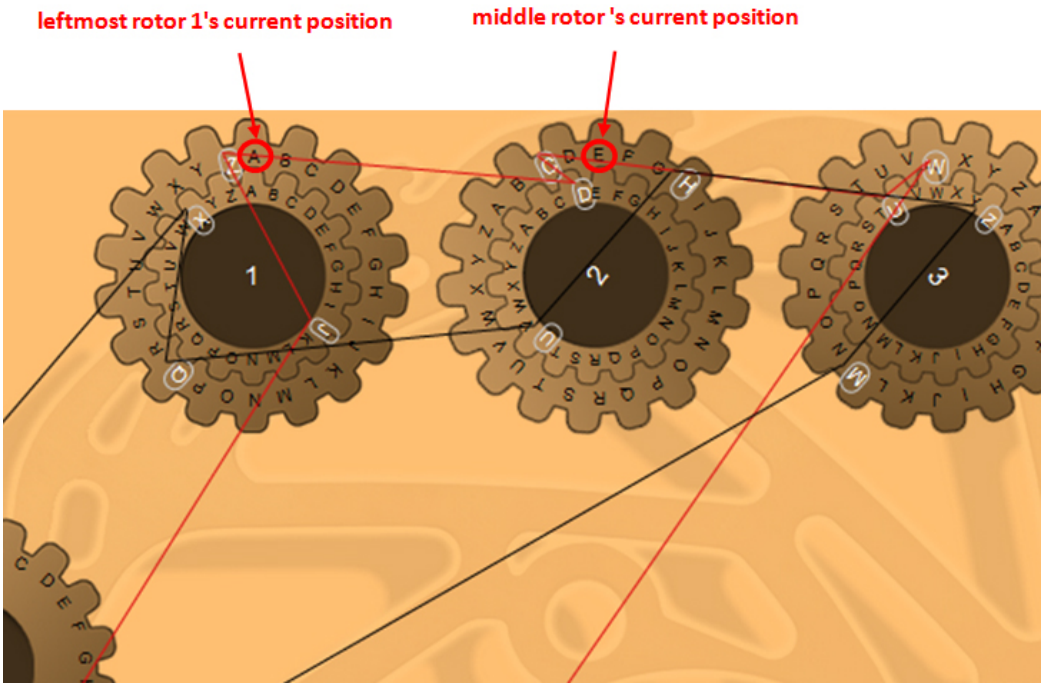

Figure 59: Enigma middle cipher rotor "double stepping"

to “F”. Figure 60 verifies that our application simulates this “double stepping” very accurately.

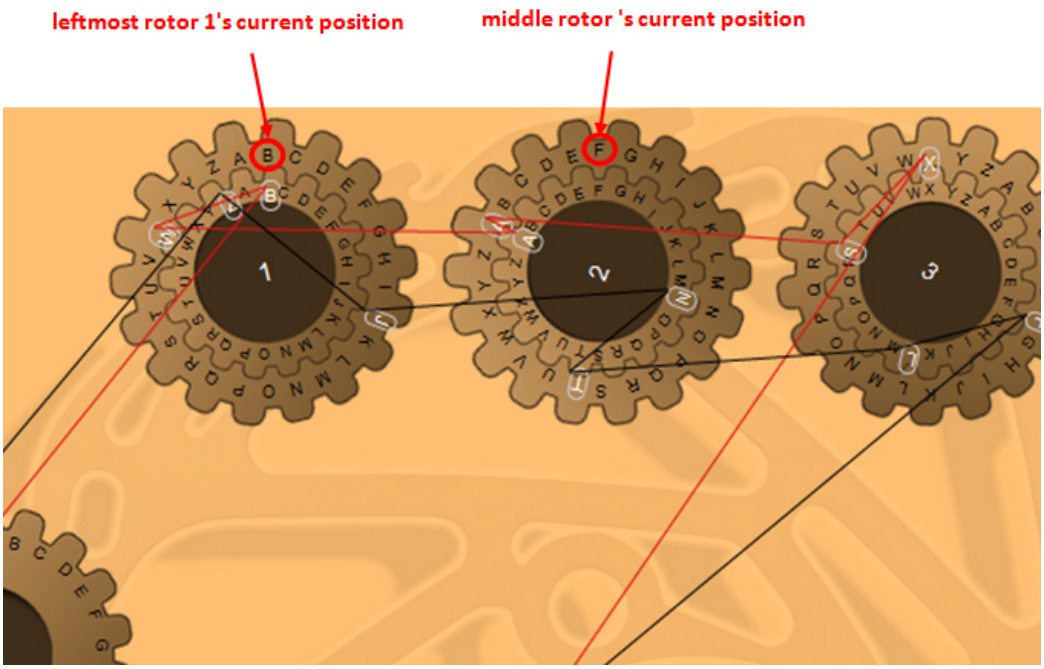

Figure 60: Enigma middle cipher rotor “double stepping”, verified

3.3.2 Simulation Visualization for Typex

When we do simulation the visualization for Typex cipher, we arrange three Typex cipher rotors at the top of the page, place leftmost cipher rotor at top left, middle cipher rotor at top middle, and rightmost cipher rotor at top right; we place two Typex stators in the middle of the page as left stator at middle left and right stator at middle right; each rotor or stator is adjusted automatically by our application so that starting letter is at 12 o’clock position. The Typex keyboard is placed at the bottom.

Recall the sample Typex key we get as we introduce visualization for Typex key setup phase, using that sample key, this means arrange Rotor 3 at top left, Rotor 2

at top middle, and Rotor 1 at top right; Rotor 4 at middle left, Rotor 5 at middle right; three cipher rotors are adjusted automatically so that letter A, letter C, and letter A are placed at 12 o'clock position for each of them; two stators are adjusted automatically so that letter M, and letter R are placed at 12 o'clock position for each of them. This example is shown in Figure 61.

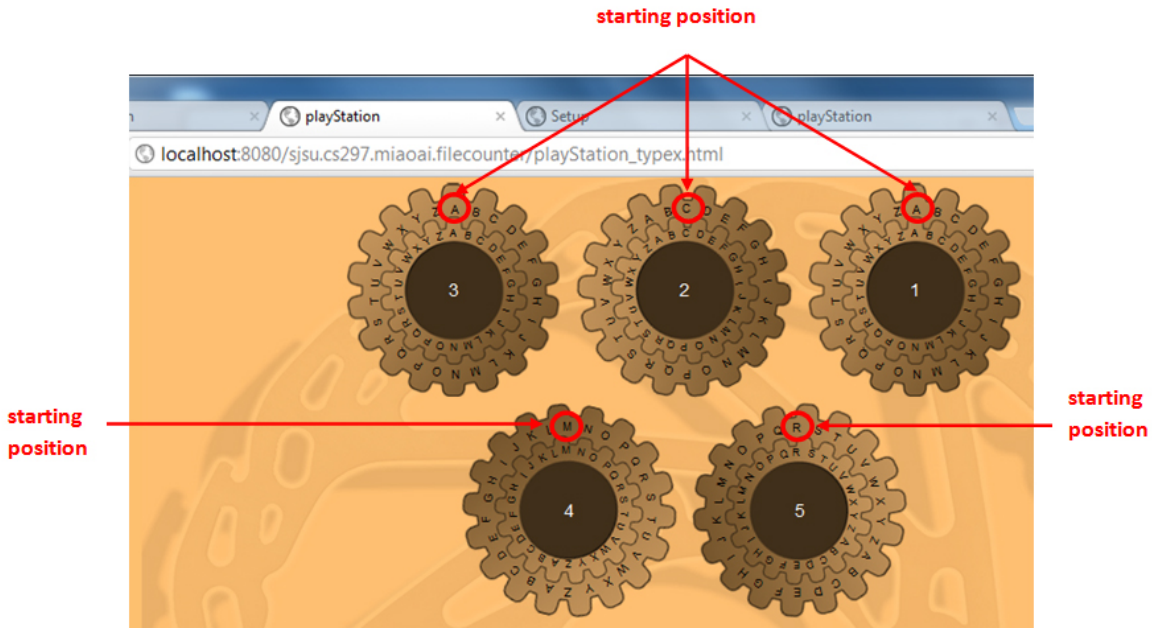

Figure 61: Visual simulation of Typex

As we already known, when Typex encrypt or decrypt begins, an electronic signal is activated and it goes through two Typex stators and three Typex cipher rotors, reaches reflector, and goes back in a inverse direction. Like Enigma, Typex also uses color lines to represent signals, red line as incoming signal, and black line as outgoing one. The key on keyboard is highlighted using red when it is pressed by user; all contacts on three cipher rotors and two stators are circled with white borders when they are touched by signals. The output is printed at outside and highlighted in red. This is illustrated at Figure 62.

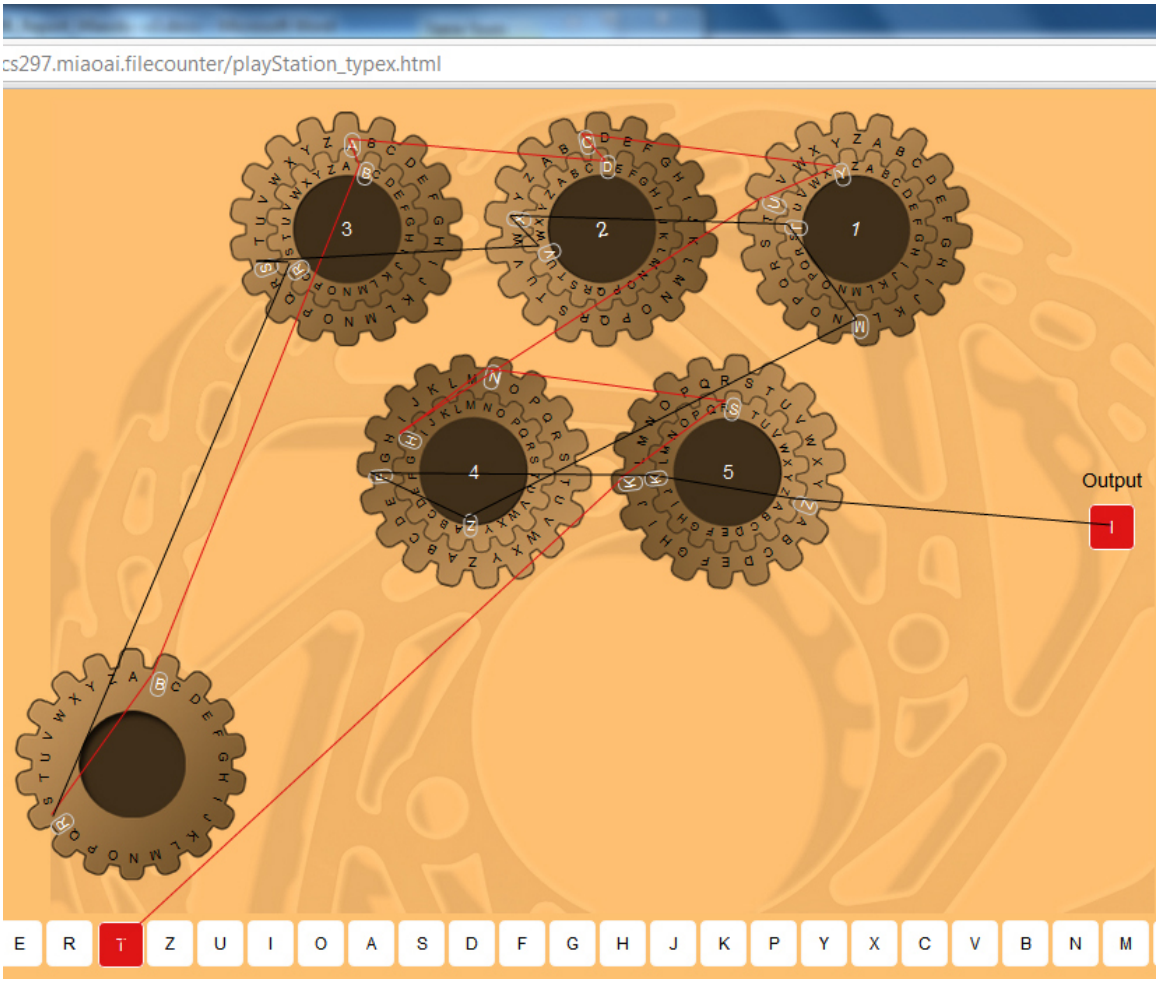

Figure 62: Visual simulation of Typex, operation begins

For example, we type a plaintext “AI MIAO CSSJSU PROJECT” in our application, and get ciphetext as “UXRAOFIZROGZFFAXMMEPZB”, this is illustrated in Figure 63.

The ciphertext has been compared with the ciphertext generated from a simulator coded using C, they are the same [12]. Figure 64 shows the ciphertext of the Typex simulator coded using C.

Use ciphertext “UXRAOFIZROGZFFAXMMEPZB” as input, with the exact same key setup, run Typex simulator again, and get “AIXMIAOXCSSJSUXPRO-

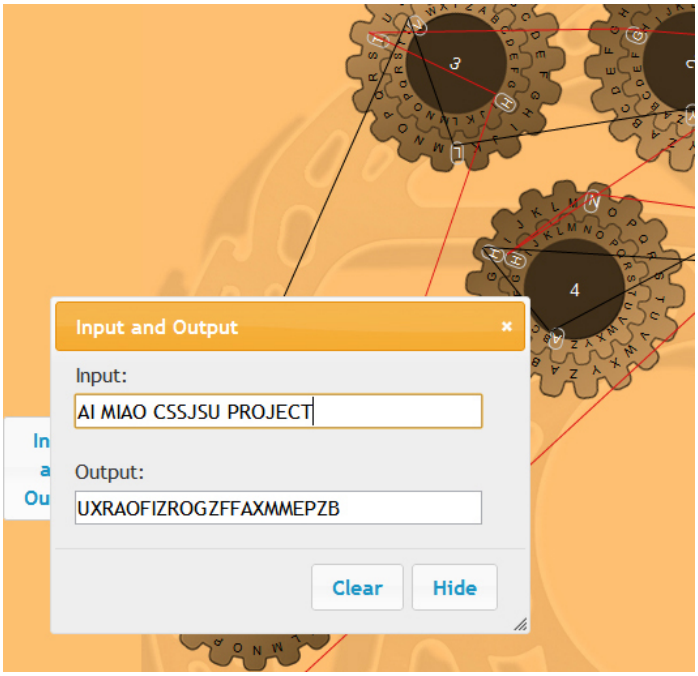

Figure 63: Visual simulation of Typex, encrypt output

```
Usage: Typex rotors orientation init infile outfile

where rotors == [L][M][R][LS][RS] rotors
            (0 thru 7, no space, no repeats)
      orientation == rotor orientations, 0 = forward, 1 = reverse
            (binary 5-tuple)
      init == initial position for [L][M][R][LS][RS] rotors
            (A thru Z, no space)
      infile == input file name

      outfile == output file name

For example: Typex 27016 01100 ZXWAK plain.txt out.txt

Note 1: Input file must contain only upper case A thru Z. Spaces are OK.


C:\MiaoAi_Files\CS298\CRYPTO_code_modified\CRYPTO_code>Typex 21034 10100 ACAMR p
laintext.txt out.txt
UXRAOFIZROGZFFAXMMEPZB
```

Figure 64: Output of a Typex simulator coded using C

JECT" as the plaintext, illustrated in Figure 65. Figure 65 confirms that our Typex visual simulator can encrypt and decrypt message correctly. More examples and test result can be found at Appendix B.

One more thing about this example we should mention is that the ciphertext "UXRAOFIZROGZFFAXMMEPZB" is decrypted as "AIXMIAOXCSSJSUXPRO-

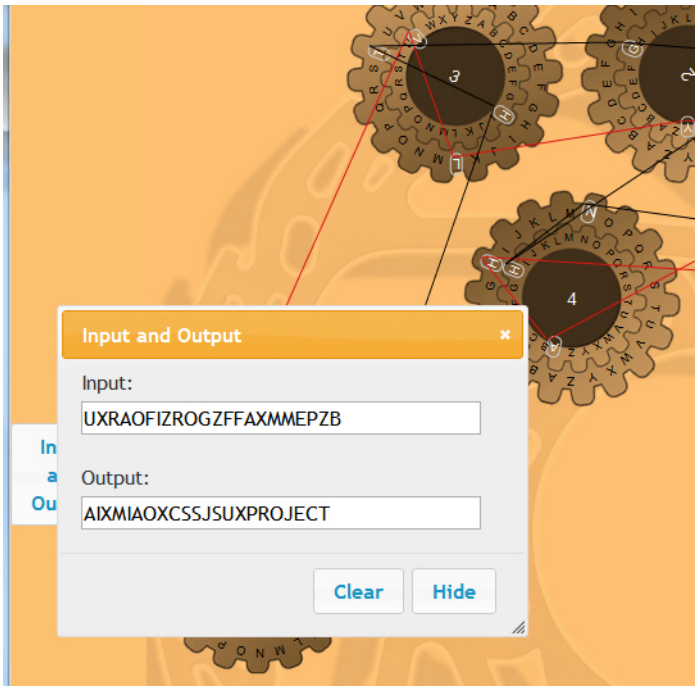

Figure 65: Visual simulation of Typex, decrypt output

JECT", instead of "AI MIAO CSSJSU PROJECT", this matches the principle about how Typex cipher encrypts and decrypts the space key we introduce at previous chapter.

3.3.3 Simulation Visualization for Sigaba

When we do the visual simulation for Sigaba cipher, we arrange five Sigaba cipher rotors at the top of the page; arrange five Sigaba index rotors in the middle of the page, and arrange five Sigaba control rotors below index rotors at the lower part of the page; each rotor is adjusted automatically by our application so that starting letter is at 12 o'clock position. The Sigaba keyboard is placed at the bottom.

Recall the sample Sigaba key we get as we introduce the visualization for Sigaba key setup phase, using that sample key, this means arrange five cipher rotors at top of the page, from left to right as Rotor 1, Rotor 9, Rotor 10, Rotor 5, and Rotor 3;

arrange five Sigaba index rotors at middle of the page, from left to right as Index Rotor 1, Index Rotor 4, Index Rotor 3, Index Rotor 5, and Index Rotor 2; arrange five Sigaba control rotors below index rotors as Rotor 6, Rotor 2, Rotor 7, Rotor 4, and Rotor 8. Five cipher rotors are adjusted automatically so that letter T, letter Z, letter U, letter E, and letter R are placed at 12 o'clock position for each cipher rotor; five control rotors are adjusted automatically so that letter A, letter H, letter N, letter M, and letter A are placed at 12 o'clock position for each control rotor; five index rotors are adjusted automatically so that number 4, number 5, number 0, number 1, and number 2 are placed at 12 o'clock position for each index rotor. This example is shown at Figure 66.

As we already know from the previous discussion, when a Sigaba operation begins, an electronic signal is activated and is divided into two signals, Signal I and Signal II. Signal I goes to control and index rotors and Signal II goes to cipher rotors. Like Enigma and Typex, Sigaba also uses color coded lines to represent signals: Signal I is represented using four lines with four different colors: green, pink, blue, and light-yellow; Signal II is represented using a red line, it is shown at the left side of the page when do encrypt operation; it is shown at the right side of the page when do decrypt operation. The key on the keyboard is highlighted in red when it is pressed by the user. All contacts on cipher, control and index rotors are circled with white borders when they are touched by signals. The output is highlighted in red and is shown at the right side of the cipher rotor bank when do encrypt operation; it is shown at the left side of the the cipher rotor bank when do decrypt operation. The signal journey for encrypt is illustrated at Figure 67; for decrypt, it is illustrated at Figure 68.

We type a plaintext: "AI MIAO CSSJSU PROJECT ZZZ" into our applica-

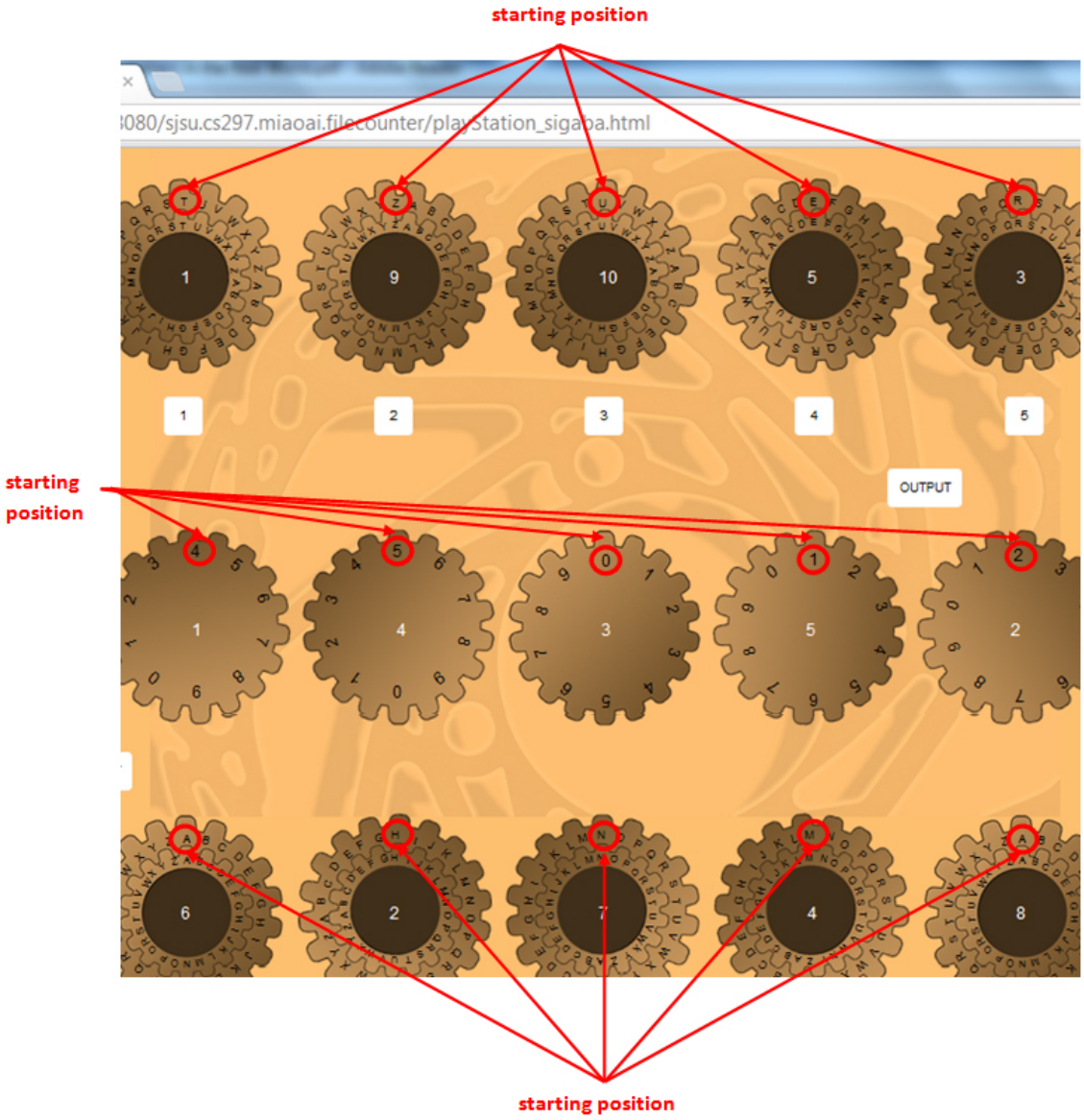

Figure 66: Visual simulation of Sigaba

tion, and get ciphertext as "DAXFVODZMGSWHRVESRPAFWYUOF", this is illustrated in Figure 69.

The ciphertext has been compared with the ciphertext generated from a Windows-based simulator and they are equal [5]. Figure 70 shows the ciphertext of the Windows-based Sigaba simulator.

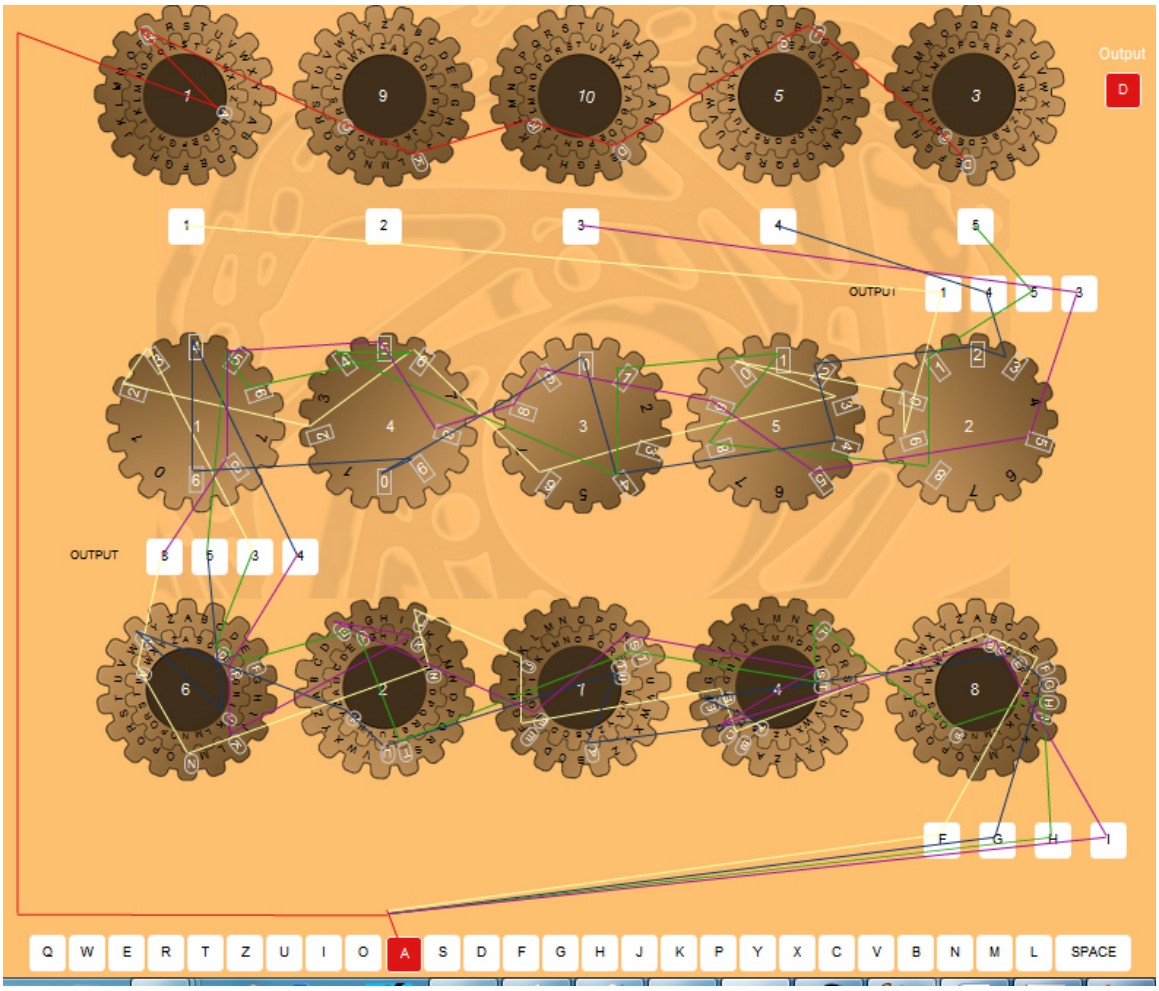

Figure 67: Visual simulation of Sigaba, encrypt operation begins

Use ciphertext “DAXFVODZMGSWHRVESRPAFWYUOF” as input, with the exactly same key but choosing decrypt mode, we run our Sigaba simulator again, and get “AI MIAO CSSJSU PROJECT XXX” as the plaintext, illustrated in Figure 71. Figure 71 confirms that our Sigaba visual simulator can encrypt and decrypt message correctly. More examples and test result can be found at Appendix C.

One more thing about this example we should mention is that the ciphertext “DAXFVODZMGSWHRVESRPAFWYUOF” is decrypted as “AI MIAO CSSJSU PROJECT XXX”, instead of “AI MIAO CSSJSU PROJECT ZZZ”, this matches

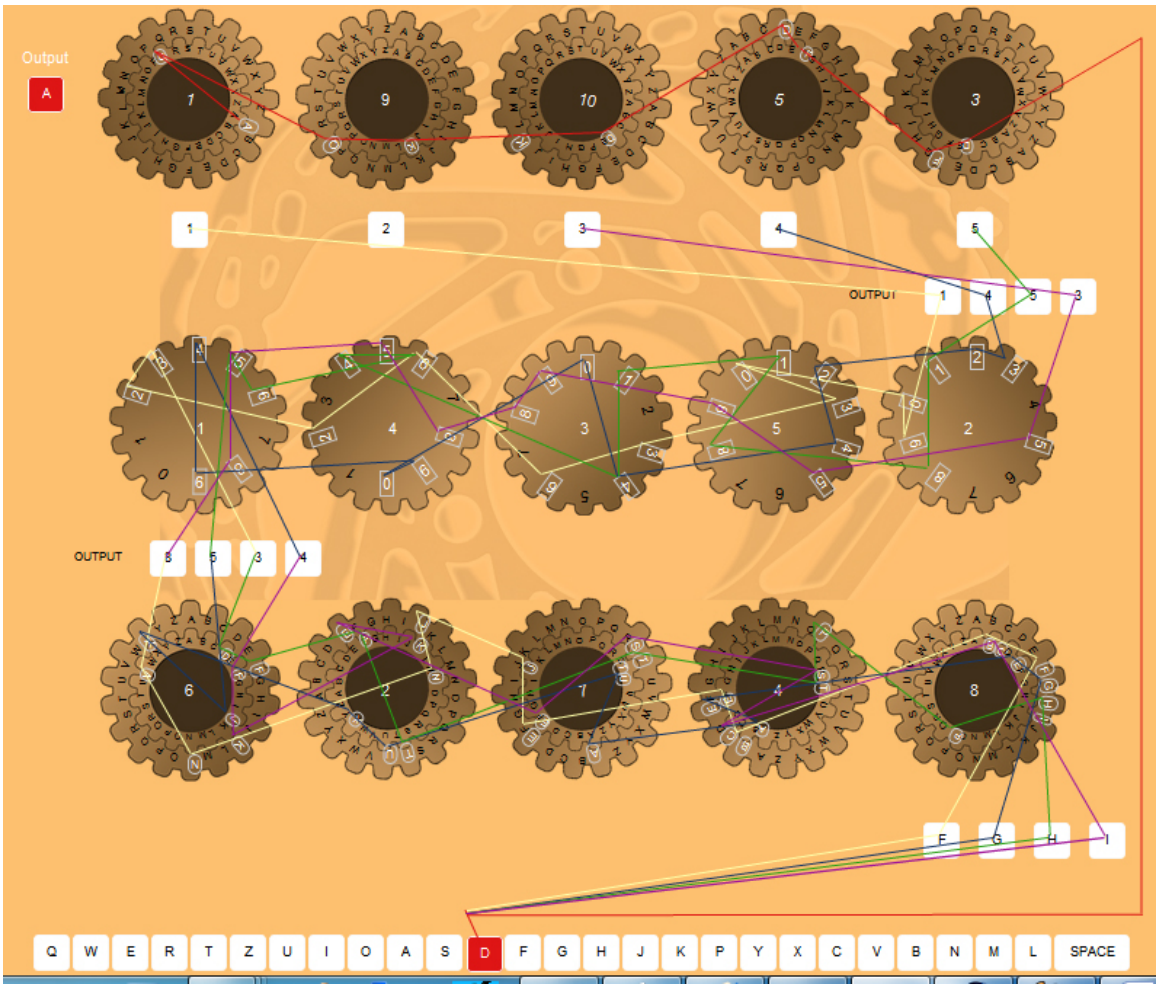

Figure 68: Visual simulation of Sigaba, decrypt operation begins

the principle about how Sigaba cipher encrypts and decrypts letter Z we introduce at previous chapter. This example also illustrates that plaintext can be encrypted and decrypted with spaces using Sigaba cipher, which makes parsing the decrypted message easier [10].

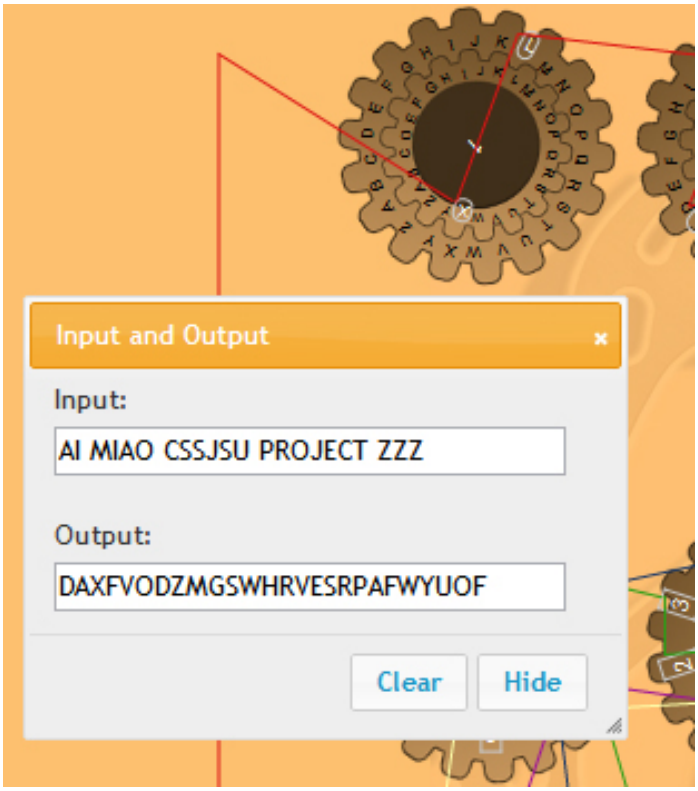

Figure 69: Visual simulation of Sigaba, encrypt output

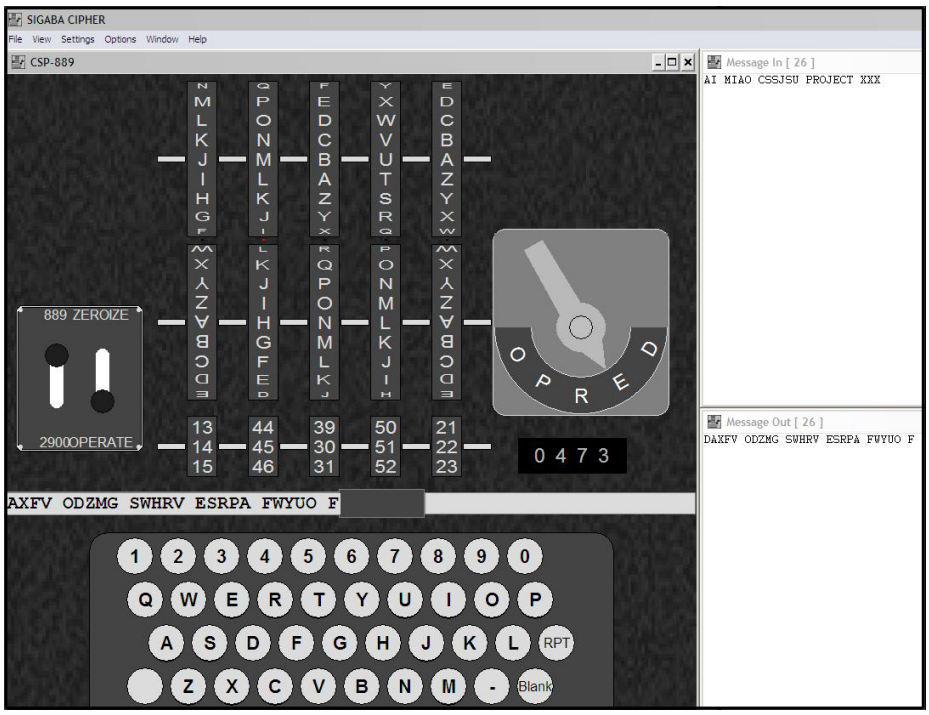

Figure 70: Ciphertext generated from a Windows-based Sigaba simulator

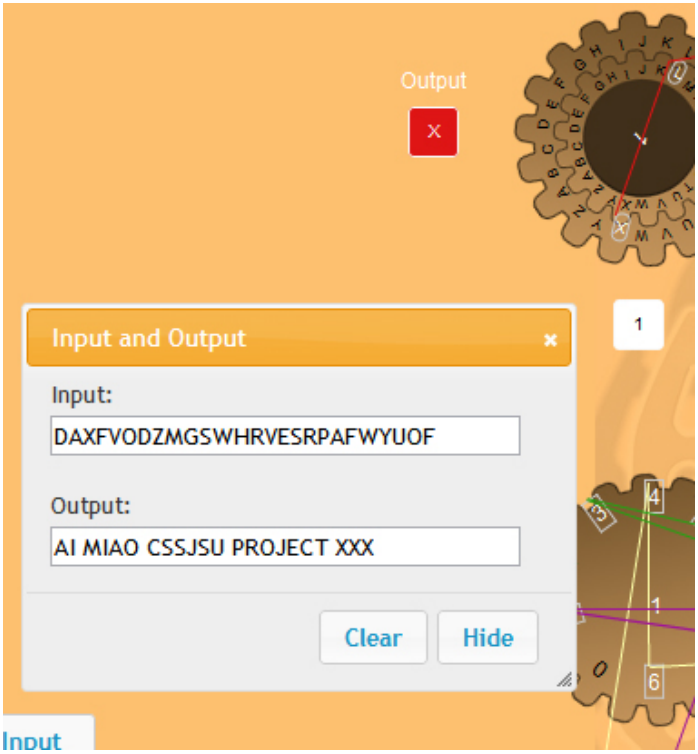

Figure 71: Visual simulation of Sigaba, decrypt output

CHAPTER 4

Conclusion and Future work

4.1 Conclusion

Classic ciphers used during WWII, like Enigma, Typex and Sigaba once played a very important role in history. For example, some people believe that the intelligence provided by Enigma decrypts may have shortened the war in Europe by a year, saving hundreds of thousands of lives [10]. As for Sigaba, it served U.S military and government during and after WWII until the 1950s, and was never broken. Therefore in a sense, Sigaba helped Americans to keep high-level communications absolutely secure [22]. Even today, those WWII ciphers are still worth our effort to learn.

In this project, we implement a web-based application that can be used as visual simulators for three classic cipher machines: Enigma, Typex and Sigaba. This application can help people who are interested in those three ciphers to understand their working principles by observing their internal structures and electronic signals journeys. It does not require any installation and has no restriction on the user's platform or operating system as long as they can access to the Internet and use Google Chrome browser.

In chapter 2, we start with introducing the background of each cipher first, followed by describing cryptographic important components for all three cipher machines, including their structures and working principles. The operations, including encrypt, decrypt, and electronic journeys descriptions for three ciphers, are introduced at the end of chapter 2.

Start from chapter 3, we begin the actual visualizations for Enigma, Typex,

and Sigaba ciphers. We start with visualizing components, followed by visualizing key setup phases for all three ciphers. During key setup visualizations, a number of examples are given, those examples generate complete sample keys can be used for Enigma, Typex, and Sigaba, respectively. Visualizing Enigma, Typex and Sigaba operations, including both encrypt and decrypt procedures are introduced at the end of chapter 3. We use those previous generated sample keys to illustrate how our application simulates signal journey(s) during each operation. Those sample keys are used to get some sample ciphertexts, and compared with ciphertexts generated from other simulators. With the same ciphertexts, we therefore confirm the correctness of our application.

4.2 Future Work

In the future, more classic cipher machines visualizations can be added into our online application, like Purple cipher machine, a non-rotor poly-alphabetic cipher that was a contemporary of the Enigma; or some modern cryptographic algorithm, like RSA, can be integrated together as well [10] [28].

There are surely more user interactive features that can be added to our application. For example, our existing application does not allow users to change any part of a key in the middle of an operation, users have to go back to key setup phase, and reset every part of a key from beginning. But this feature could be useful and convenient if a user just wants to change a part of key during operation. Some other useful features could be a "zoom in" button, which allows user to zoom in a particular component, a rotor, for example, to exam how signal(s) goes through this component with a clear view.

LIST OF REFERENCES

[1] M. Rejewski, An Application of the Theory of Permutations in Breaking the Enigma Cipher, *Applicaciones Mathematicae*, 1980

[2] F. Spiess, Three Rotor Enigma Simulation,
`http://enigmaco.de/_fs/index-enigma.html`

[3] W. O. Chan, *Cryptanalysis of Sigaba*, Masters Report, Department of Computer Science, San Jose State University, 2007

[4] H. E. Kwong, *Cryptanalsis of the Sigaba Cipher*, Masters Report, Department of Computer Science, San Jose State Univeristy, 2008

[5] G. Sullivan and F. Weierud, Geoff's Crypto page,
`http://www.hut-six.co.uk`

[6] Enigma machine - Wikipedia, the free encyclopedia,
`http://en.wikipedia.org/wiki/Enigma_machine`

[7] B. D. Brown, Enigma - German Machine Cipher - "Broken" by Polish Cryptologists,
`http://math.ucsd.edu/~crypto/students/enigma.html`

[8] Rau Antiques Sales Very Rare WWII Enigma Cipher Machine,
`http://www.extravaganzi.com`

[9] T. Sale, The Enigma cipher machine,
`http://www.codesandciphers.org.uk/enigma/index.htm`

[10] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley, 2007

[11] D. Rijmenants, The German Enigma Cipher Machine,
`http://users.telenet.be/d.rijmenants/en/enigma.htm`

[12] K. Chang, *Cryptanalysis of Typex*, Masters Report, Department of Computer Science, San Jose State Univeristy, 2012

[13] Enigma rotor details - Wikipedia, the free encyclopedia,
`http://en.wikipedia.org/wiki/Enigma_rotor_details`

[14] M. Stamp, *Information Security: Principles and Practice*, second edition, Wiley, 2011

[15] Plugboard,
http://enigma.wikispaces.com/Plugboard

[16] Enigma,
http://www.cryptool-online.org/index.php

[17] TYPEX,
http://www.jproc.ca/crypto/typex.html/

[18] Typex - Wikipedia, the free encyclopedia,
http://en.wikipedia.org/wiki/Typex

[19] G. Sullivan, TYPEX,
http://www.hut-six.co.uk/typex/index.html

[20] K. Chang, personal correspondence

[21] L. Kruh and C. A. Deavours, The Types Cryptography, *Cryptologia*, Vol. 7, pp. 145–167, 1983

[22] SIGABA (ECM),
http://www.cryptomuseum.com/crypto/usa/sigaba/index.htm/

[23] R. Pekelney, SIGABA - ECM MARK II,
http://www.jproc.ca/crypto/ecm2.html

[24] SIGABA - Wikipedia, the free encyclopedia,
http://en.wikipedia.org/wiki/SIGABA

[25] National Cryptologic Museum, 16 November 2007,
http://yank.to/Photos/2007/20071116_National_Cryptologic_Museum

[26] W. O. Chan and M. Stamp, SIGABA: Cryptanalysis of the Full Keyspace, *Cryptologia*, Vol. 31, pp. 201–222, 2007

[27] L. Safford and D. Seiler, Control circuits for electric coding machines, United States Patent Number 6,175,625, washington, DC, 2001

[28] RSA (algorithm) - Wikipedia, the free encyclopedia,
http://en.wikipedia.org/wiki/RSA

APPENDIX A

Enigma Verification

In order to verify the correctness of our application on simulating Enigma cipher, a couple of larger tests have been tested for both encryption and decryption operations. We use the following notation for key used for Enigma test case:

Enigma key: [leftmost cipher rotor id, starting position][middle cipher rotor id, starting position][rightmost cipher rotor id, starting position][reflector rotor id][stecker setting]

Test case 1

We use Enigma key: [2, A][1, Z][3,T][1][A-A, B-B, C-C, D-F, E-E, G-G, H-H, I-I, J-J, K-K, L-L, M-M, N-N, O-O, P-P, Q-Q, R-R, S-S, T-T, U-U, V-V, W-W, X-X, Y-Y, Z-Z, F-D] to encrypt following message, the length of this message is 200 letters.

1st plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAO
NETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMA
ONETWO

1st ciphertext:

LXBMNWFPXUPTFTJYHWTUYBOHCANAHZEBYXPMJCLJJNBGNAFA
LTKRBTKVSDYMKFXNWCCBXZKEYFHANFEGFDYLHDFGYIFZPWHWTSL
RWUYKRBBHGKYYBCERLECXXHKJBFZIPVYKQWRQYHSIZNZEUINIJETY

JRELUFOIVZYBFCPHLAOLVKPSQUBPWDSZBWBINZGOGPGWQLBU

We then used the same Enigma key to decrypt the ciphertext, the result is the same plaintext as we expected.

Test case 2

We use Enigma key: [3, X][1, W][4, A][2][A-A, B-B, C-C, D-F, E-E, G-G, H-H, I-I, J-J, K-K, L-L, M-M, N-N, O-O, P-P, Q-Q, R-R, S-S, T-T, U-U, V-V, W-W, X-X, Y-Y, Z-Z, F-D] to encrypt following message, the length of this message is 100 letters.

2nd plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWO

2nd ciphertext:

GVDVUMOCTIUCYTNJQLTCWHALWYRFAKASJHWJQDBPUWQMZVC
PCRZGAKCVNSUDGRJVMBHLRZRDLTGVXEEDJSIQNOBPZWRHOEJKUCP
F

We then used the same Enigma key to decrypt the ciphertext, the result is the same plaintext as we expected.

Test case 3

We use Enigma key: [1, H][2, D][3, X][1][A-A, B-B, C-C, D-F, E-E, G-G, H-H, I-I, J-J, K-K, L-L, M-M, N-N, O-O, P-P, Q-Q, R-R, S-S, T-T, U-U, V-V, W-W, X-X, Y-Y, Z-Z, F-D] to encrypt following message, the length of this message is 500 letters.

3rd plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAO
NETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMA
ONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAM
AONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIA
MAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWO
IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETW
OIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONET
WOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWO

3rd ciphertext:

WTERDZQBBXXFJPXFPHRGQXEOFMWGZNNWFNCICVJDQRVFJMVF
DEOZTYMQFPIBLTDNHCXDLNWSTONLTYVLYMAFUJLGSPFFTPBFBOFY
KKPUHKXIGKXYNNNUCGPJKMZZNMOGEXAXFZFKCJHBMQXNRMJSBZ
WEQMAUHWIEBZORKBYNJABIXWXMZXHWGTWFZFZGRTKILSCSWWM
TSWYCMYFLDUNZJCLEFQJCROKUMCKBLCDVKESLKNSPQOJEBCNUXG
QUGXRDDEDJPCWMSXTBNVXPYMMGBLSHRIZESUBIIRLJPDAOYEONOI
RYWTRMVZBQXEJWTQHGBKZVDWAYEZFUPSLVXMZWJLLNCKGCVGBO
WHPWQMYWEEGVKPLQSMJCWEUUNGGCACUOWGQNYZCYTJZEHGMC
VBEBQXXQECFMNVYSQNUHJBQZPFOWTPKJVEPIHRRGZDXHXZKOLIJZ
WODWYLAYXUNJCXRPNDCUQJFLZMXHPBEGVDOXURXYXFTSCRLJ

We then used the same Enigma key to decrypt the ciphertext, the result is the same plaintext as we expected.

APPENDIX B

Typex Verification

B.1 Justification for correctness of Typex reverse rotor

We implement Typex cipher and stator rotor in reverse orientation the same way as Kelly Chang did in her master thesis "Cryptanalysis of Typex". In her C code, she reverses a rotor by reversing its permutation. For example, Figure B.72 shows her code for reversing leftmost cipher rotor.

```
// reverse Left rotor
if (x == 0) {
    for (i = 0; i < 26; ++i) {
        for (j = 0; j < 26; ++j) {
            newrotor[i][j] = L[i][(26 - j) % 26];
        }
    }
    for (i = 0; i < 26; ++i) {
        for (j = 0; j < 26; ++j) {
            L[i][j] = newrotor[i][j];
        }
    }
}
```

Figure B.72: Reversing Typex leftmost cipher rotor, coded in C

We confirm its correctness by printing out a permutation for the left rotor before and after reversing as illustrated at Figure B.73 and Figure B.74.

```
C E G I K B O Q S W U Y M X D H V F Z J L T R P N A
```

Figure B.73: The permutation of left rotor, before reversed

```
C A N P R T L J Z F V H D X M Y U W S Q O B K I G E
```

Figure B.74: The permutation of left rotor, after reversed

As can be seen from Figure Figure B.73 and Figure B.74, the original permutation is reversed after being processed by the code. Therefore we implement our application

using the same way as the above code, and we run a number of test cases as shown in the next section to confirm the output from our application matches the output from Kelly Chang's Typex simulator.

B.2 Typex test cases

In order to verify the correctness of our application on simulating Typex cipher, a couple of larger tests have been tested for both encryption and decryption operations. We use the following notation for key used for Typex test case:

Typex key: [leftmost cipher rotor id, starting position, orientation][middle cipher rotor id, starting position, orientation][rightmost cipher rotor id, starting position, orientation][left stator id, starting position, orientation][right stator id, starting position, orientation]

Test case 1

We use Typex key: [3, A, 1][2, C, 0][1, A, 1][4, M, 0][5, R, 0]to encrypt following message, the length of this message is 500 letters.

1st plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAO
NETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMA
ONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAM
AONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIA
MAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWO
IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETW

OIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONET
WOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWO

1st ciphertext:

TUSMIZTQKSHLHLJKXFQFUJQSYVRDTGBTNXTPJDVGZECXXFNPE
WKZDVICAJZGWLSRHKSMFBLMURTTCXUKRNADNLXVDCVVGUDDSAP
UGNTIEHGZBJEHCWFUMJFNTWCCRXEPCAFCWAGTRSLKRUTTPZCSNC
GWUFAZKQNKXUBPDMKMTCXJTZUUHCKRFSYVUXIVCESBVWZQUYGV
RGBSGXRNNMBJTAPZEDBYSCZYOSIUAECXKTRAQVHLZJIQAXFGOSHTC
WNZRCYXQKYIDNQISFYXVJPLZCDUIKZFHEEAYMTNCALURQVYDBDON
WOYJOCFJLHOLVANURRYAROAQGVQIHJOOFECQIQWWUOIMGNEYLAF
YJRTTDLZNUZFSROBGEZFQYQWYPLSIVDIKKWHCFYMEXVWYIPKFUOJ
GBBWBXSJADXRBWRVKNYDYBCHBAZRDBSPSDFVSHXXELLEKQTMXPY
FRDIDTKAXWXJBUUTWIKVNHJRQBEYYPMZTXJRAIE

We then used the same Typex key to decrypt the ciphertext, the result is the same plaintext as we expected.

Test case 2

We use Typex key: [2, Z, 0][3, K, 0][5, M, 0][4, L, 0][1, A, 1]to encrypt following message, the length of this message is 200 letters.

2nd plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAO

NETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMA
ONETWO

2nd ciphertext:

KUNVVTUWNIJKHLMXRBCNQSPSDPCCHIEPWFPVYGEDPHBBWSIE
XQQBCCVCBVLQKNOSCRUDBNWDZPDLJCDSUZJZIPJZFBNGZJMRBNQP
TCCSJETKGVPDSYBISMXDJGDRNKMQVTOIFHMEFXCBUSCUAPQQTLS
OTZEDGXXZLIWPNGYGBPMPFCAMFWZDVPCXWBAQYGQBNXLVZVMR

We then used the same Typex key to decrypt the ciphertext, the result is the same plaintext as we expected.

Test case 3

We use Typex key: [4, G, 0][3, A, 1][5, D, 0][1, R, 0][2, P, 0]to encrypt following message, the length of this message is 100 letters.

1st plaintext:

IAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONE
TWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAONETWOIAMAON
ETWO

1st ciphertext:

TIHCEKILCKCFLNWUWXFURGPTUSHMVMTHAYKFWQTGSSLCJOUG
ODSRAZIUGLVHQXPWEAHDVCNKPUGKTEPMMWVLMASDTFVJYFEYIH
QQ

We then used the same Typex key to decrypt the ciphertext, the result is the same plaintext as we expected.

APPENDIX C

Sigaba Verification

C.1 Justification for correctness of Sigaba reverse rotor

We implement Sigaba cipher and control rotor in reverse orientation using the same idea as Geoff Sullivan did for his Sigaba simulator. In one of the email from Geoff Sullivan, he explains that "A reverse rotor presents a different wiring pattern to the machine and also steps in reverse, even though it is stepping forwards physically." According to Geoff, he translated the wiring ([LEFT] and [RIGHT] for normal and reversed) similar to the java code in Figure C.75 and Figure C.76.

```
class CipherRotor extends Rotor {
    int cipherRotor[][] = new int[2][26];

    CipherRotor(int wiringNum) {    // Constructor for Cipher Rotors.

    int i;

        for(i = 0 ; i < 26 ; i++) {
            cipherRotor[LEFT][i] = WIRING[wiringNum][i] - (int) 'A';
            cipherRotor[RIGHT][cipherRotor[LEFT][i]] = i;
            }
```

Figure C.75: Cipher rotor coded in JAVA

```
class ControlRotor extends Rotor {
    int controlRotor[][] = new int[2][26];

    ControlRotor(int wiringNum) {    // Constructor for Control Rotors
        int i;

        for(i = 0 ; i < 26 ; i++) {
            controlRotor[LEFT][i] = WIRING[wiringNum][i] - (int) 'A';
            controlRotor[RIGHT][controlRotor[LEFT][i]] = i;
            }
```

Figure C.76: Control rotor coded in JAVA

After several back and forward emails with Geoff, we are able to successfully implement Sigaba cipher and control rotor in reverse orientation using the same idea as above, in a slight different way. And we run a number of test cases as shown in

the next section to confirm the outputs from our application match the outputs from Geoff Sullivan's Sigaba simulator.

C.2 Sigaba test cases

In order to verify the correctness of our application on simulating Sigaba cipher, a couple of larger tests has been tested for both encryption and decryption operations. We use the following notation for key used for Sigaba test case:

Sigaba key: [first cipher rotor id, starting position, orientation][second cipher rotor id, starting position, orientation][third cipher rotor id, starting position, orientation][fourth cipher rotor id, starting position, orientation][fifth cipher rotor id, starting position, orientation][first control rotor id, starting position, orientation][second control rotor id, starting position, orientation][third control rotor id, starting position, orientation][fourth control rotor id, starting position, orientation][fifth control rotor id, starting position, orientation][first index rotor id, starting position][second index rotor id, starting position][third control rotor id, starting position][fourth control rotor id, starting position][fifth control rotor id, starting position][operation mode]

Test case

We use Sigaba key: [1, T, 0][9, Z, 0][10, U, 0][5, E, 0][3, R, 0][6, A, 1][2, H, 0][7, N, 0][4, M, 0][8, A, 1][1, 4][4, 5][3, 0][5, 1][2, 2][Encryption] to encrypt following message, the length of this message is 50 letters.

plaintext:

IAMAONETWOIAMAONETWOIAMAONETWO

ciphertext:

EVYAMJODSNVDGKTLQXVWOYVFHHHYAF

We then used the Sigaba key: [1, T, 0][9, Z, 0][10, U, 0][5, E, 0][3, R, 0][6, A, 1][2, H, 0][7, N, 0][4, M, 0][8, A, 1][1, 4][4, 5][3, 0][5, 1][2, 2][Decryption] to decrypt the ciphertext, the result is the same plaintext as we expected.